目　录

第一章 Simulink 快速入门

1.1 Simulink 简介

近几年,Simulink 已经在学术和工业等领域得到了广泛的应用,用它可以进行动态系统的建模和仿真,也可以很随意地建立各种模型. Simulink 仿真是交互式的,可以很随意地改变模型的参数并且马上就可以看到改变参数后的结果. MATLAB 中的分析与可视化工具多种多样并且易于操作,所以用户可以对仿真的结果进行分析并且使之可视化. 在这样的环境中你会发觉用 Simulink 进行建模和仿真是一件很有趣的事情. 这一环境激励着你不断地提出新问题,并对问题进行建模,最后得到仿真结果.

通过应用 Simulink 建模与仿真,你可以超越理想的线性模型而去探求更为现实的非线性模型,比如现实世界中摩擦、空气阻力、齿轮的滑动等自然现象. Simulink 会使你的计算机变成一个实验室,以用来对各种现实中不可能存在或现实中恰恰相反的系统进行建模和仿真. 不管是汽车离合器的动作,飞机机翼的抖动,还是货币供给对经济的影响等都可以进行建模和仿真.

1.1.1 什么是 Simulink

Simulink 是一个用来对动态系统进行建模、仿真和分析的软件包. 它支持线性和非线性系统,连续和离散时间模型,或者是两者的混合. 系统还可以是多采样率的,比如系统的不同部分拥有不同的采样率.

对于建模,Simulink 提供了一个图形化的用户界面(GUI),可以用鼠标点击和拖拉模块的图标建模. 通过图形界面,可以像用铅笔在纸上画图一样画模型图. 这是以前需要用编程语言明确地用公式表达微分方程的仿真软件包所远远不能相比的. Simulink 包括一个复杂的由接受器、信号源、线性和非线性组件以及连接件组成的模块库,当然也可以定制或者创建用户自己的模块.

所有模型是分级的,因此可以通过自上而下或者自下而上的方法建立模型. 可以在最高层面上查看一个系统,然后通过双击系统中的各个模块进入到系统的低一级层面以查看到模型的更多的细节. 这一方法提供了一个了解模型是如何组成以及它的各个部分是如何相互联系的方法.

定义完一个模型以后,就可通过 Simulink 的菜单或者在 MATLAB 的命令窗口输入命令对它进行仿真. 菜单对于交互式工作非常方便,而命令行方式对于处理成批的仿真比较有用(例如,你在进行 Monte Carlo 仿真时想使参数遍历某一范围的值). 使用 Scopes 或者其它的显示模块,可以在运行仿真时观察到仿真的结果. 另外,还可以在仿真时改变参数并且立即就可看到有什么变化. 仿真的结果可以放在 MATLAB 的工作空间(workspace)中以待进一步的处理或者可视化.

模型分析可使用的工具包括可直接通过命令行方式调用的线性化和整理(trimming)

工具，MATLAB 的其它各种工具，以及所有应用程序工具箱. 因为 MATLAB 和 Simulink 是集成在一起的，所以用户可以在任何环境的任意点对用户的模型进行仿真、分析或修改.

1.1.2 Simulink 实时工作环境的作用及其主要特点

Simulink 实时工作环境(Real-Time Workshop®)自动地直接从 Simulink 的模块图生成 C 语言代码. 这将允许连续、离散时间或者混合系统的模型可以运行于各种计算机平台，其中包括实时硬件，但 Simulink 是必不可少.

1.1.2.1 实时工作环境的作用

1) 快速建模　　作为一个快速建模工具，实时工作环境使得用户可以快速实现自己的设计，而不用手工编写长长的代码然后进行调试. 控制、信号处理和动态系统的算法可以通过开发图形化的 Simulink 模块图，并且自动生成 C 语言源码来实现.

2) 嵌入式实时控制　　一旦一个系统已经用 Simulink 设计出来，就可生成实时控制器或数字信号处理器的代码，然后可对代码进行编译、链接，最后装载到目标处理器中. 实时工作环境支持 DSP 板，嵌入式控制器，以及多种用户和商业开发的硬件.

3) 实时仿真　　对循环中硬件仿真，可以为整个系统或指定的分系统创建和执行代码. 典型的应用包括训练仿真器，实时模型验证和测试.

4) 单机仿真　　单机仿真可以在你的主机上直接运行或者传送到另外的系统上以远程方式执行. 由于时间历史数据被以二进制或 ASCII 文件保存在 MATLAB 中，可以很容易地被装入 MATLAB 中以待进一步的分析或图形显示.

1.1.2.2 实时工作环境的主要特点

实时工作环境具有一系列复杂的能力和特性以提供实现各种应用的灵活性.

1) 自动代码生成以处理连续时间、离散时间和混合系统.

2) 优化的代码以保证快速执行.

3) 控制框架结构应用程序接口(API)自动地使用定制的 make 文件来创建和下载 object 文件到目标硬件上.

4) 可移植的代码使其应用环境更加广泛.

5) 简明、可读并具有详细注释的代码使得维护非常简单.

6) 从 Simulink 下载到外部硬件上的交互参数使系统在工作状态下很容易调整.

7) 一个菜单驱动的图形用户界面使得软件的使用非常容易.

1.1.2.3 实时工作环境支持的目标环境

1) 使用 TI C30/C31/C40 DSP 的 dSPACE DS1102, DS1002, DS1003;

2) VxWorks, VME/68040;

3) 带有 Xycom, Matrix, Data Translation，或者计算机板 I/O 设备和 Quanser Multiq 板的基于 486 PC 的系统.

1.1.2.4 实时工作环境 Ada 扩展

Simulink 实时工作环境 Ada 扩展能自动地直接从 Simulink 的模块图生成 Ada 代码. 这将使得连续、离散时间或者混合系统的模型可以运行于各种计算机平台,其中包括实时硬件,但 Simulink 也是必不可少.

Simulink 实时工作环境 Ada 扩展为下述 Ada 83 编译器提供了转键:UNIX 平台的 Rational VADS,微软 Windows 专业版的 Thomson ActivAda,微软 Windows NT 的 Thomson ActivAda.

1.1.3 模块集

与 MATLAB 及其应用工具箱相似,MathWorks 公司为 Simulink 提供了模块集. 模块集是 Simulink 模块的集合,它们被分成不同的组.

1.1.3.1 DSP 模块集

DSP 模块集是为了快速设计和仿真基于 DSP 的设备和系统,而对 Simulink 进行的扩展. 有了 DSP 模块集,Simulink 提供了一个基于模块图的仿真和评价信号处理算法的直观工具. 它的图形化编程环境使得工程技术人员很容易地就能创建、修改和定型 DSP 设计.

DSP 模块集的应用包括通信系统、计算机外设、语音处理、汽车和飞机控制、电子医疗设备的设计和分析. 它对于时域和频域的算法都是非常理想的,包括诸如自适应噪声消除等问题.

1.1.3.2 定点运算模块集

定点运算模块集与 Simulink 一起使用,它包括一个对标准的 Simulink 模块库进行扩展的模块图组件的集合. 有了这一系列的模块,就可以用定点算法创建离散的动态系统. 这样,Simulink 就可对诸如控制系统和时域滤波之类的应用中常会遇到的定点运算系统的效果进行仿真.

定点运算模块集允许用户在一个方便高效的环境中对定点运算的效果进行仿真. 定点运算模块集提供的新模块包括:

1) 加法和减法;
2) 乘法和除法;
3) 求和;
4) 增益和常量;
5) 浮点和定点信号之间的转换;
6) 一维和二维查找表;
7) 逻辑运算;
8) 关系运算;
9) 定点信号的转换/饱和;
10) 两值之间切换;

11）延迟；

12）Δ 逆运算；

13）监视信号.

信号转换模块可以使信号在浮点和定点之间转换. 使用转换模块，可以创建由 Simulink 的标准模块库组件和定点模块组成的 Simulink 模块图.

可以用标准的 Simulink 模块创建模型，并且用定点模块对控制器进行建模. 数据范围模块提供了进行仿真时模块图的任一点所能取得的最大和最小值.

定点模块使用户可以用无符号整数或 8 位、16 位、32 位字长的二进制补码进行建模. 同一模块图中可以使用不同字长. 定点运算的数值是通过在定点模块中指定二进制点的位置而得到的. 在仿真过程中，数据类型可以改变，从而立即得到不同字长、二进制点位置、四舍五入、溢出检查的结果.

这一模块集的另一强大的功能是能够确定二进制点的位置，使得数据精度最大而又不至于溢出. 通过数据范围模块，可以指定二进制点的位置为一个合适的值.

定点模块要求 Windows 或 UNIX 操作系统下的 MATLAB 5.3 和 Simulink 3.0.

1.1.3.3 非线性控制设计模块集

非线性控制设计(NCD)模块集提供了基于时域的、鲁棒的、非线性的控制设计. 在 Simulink 中控制器是通过模块图来设计的. 可以为给定的输出信号选择一系列可调的模型参数，并且图形化地设置其时间响应约束. 通过调节选定的模型参数自动地使用连续的仿真和优化方法.

1.1.3.4 电源系统模块集

电源系统模块集使得工程技术人员可以建立模型来仿真电源系统. 该模块集使用 Simulink 环境，可以通用点击或拖拉来建立模型. 不仅可以快速画出电路的拓扑结构，而且电路的分析可以包括其与机械、热、控制及其它学科的相互作用. 这是由于仿真的电部分与 Simulink 的庞大的模型库是相互作用的. Simulink 使用 MATLAB 作为计算引擎，MATLAB 其它工具箱也可用于设计.

该模块集库中包含典型电源设备，如变压器、电缆、电机、电源电子设备等.

1.2 Simulink 快速入门

1.2.1 运行一个示例模型

Simulink 提供的一个很有趣的例子是对房子的热力学进行建模. 可按下列步骤运行这一示例.

启动 MATLAB；通过在 MATLAB 的命令窗口输入 thermo 命令运行示例模型. 这一命令启动 Simulink 并且创建一个模型窗口，模型窗口中包含如图 1.1 所示的模型.

当你打开这一模型时，Simulink 同时打开一个显示器(Scope)模块，包含两个名称分别为“Indoor vs. Outdoor Temp”和“Heat Cost($)”的曲线图，如果没有自动打开，可双

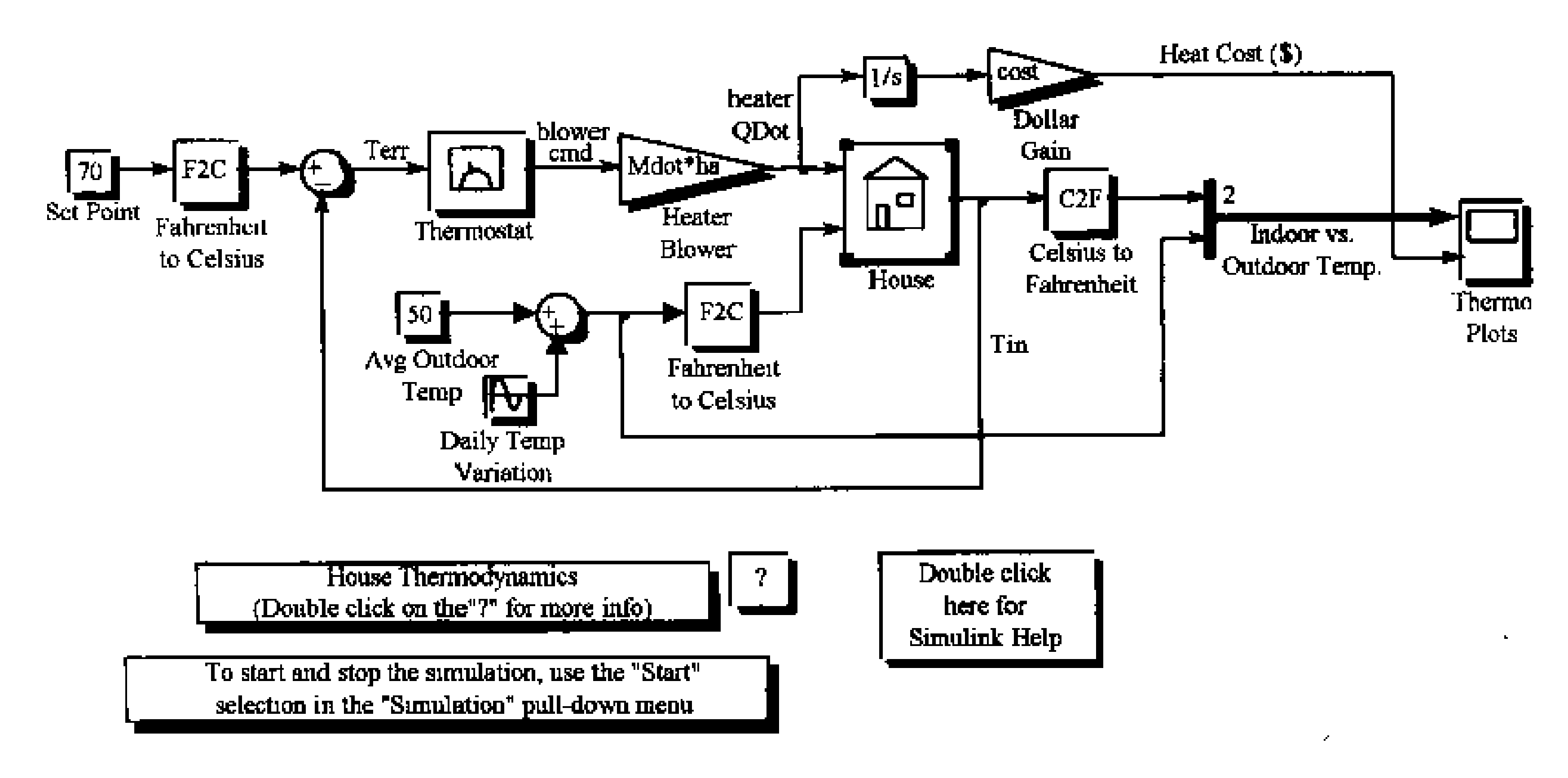


图 1.1　示例模型

击模型图中的“Thermo Plots”显示模块.Scope 显示窗口如图 1.2 所示.

要启动该仿真,下拉主菜单中的“Simulation”,并选择“Start”菜单项,或者按“Ctrl”和“T”键,也可以按工具条中的“开始按钮”.当仿真运行时,室内和室外的温度显示在显示器模块窗口的“Indoor vs. Outdoor Temp”内,同时累计的热量消费显示在 Scope 模块窗口的“Heat Cost($)”内.

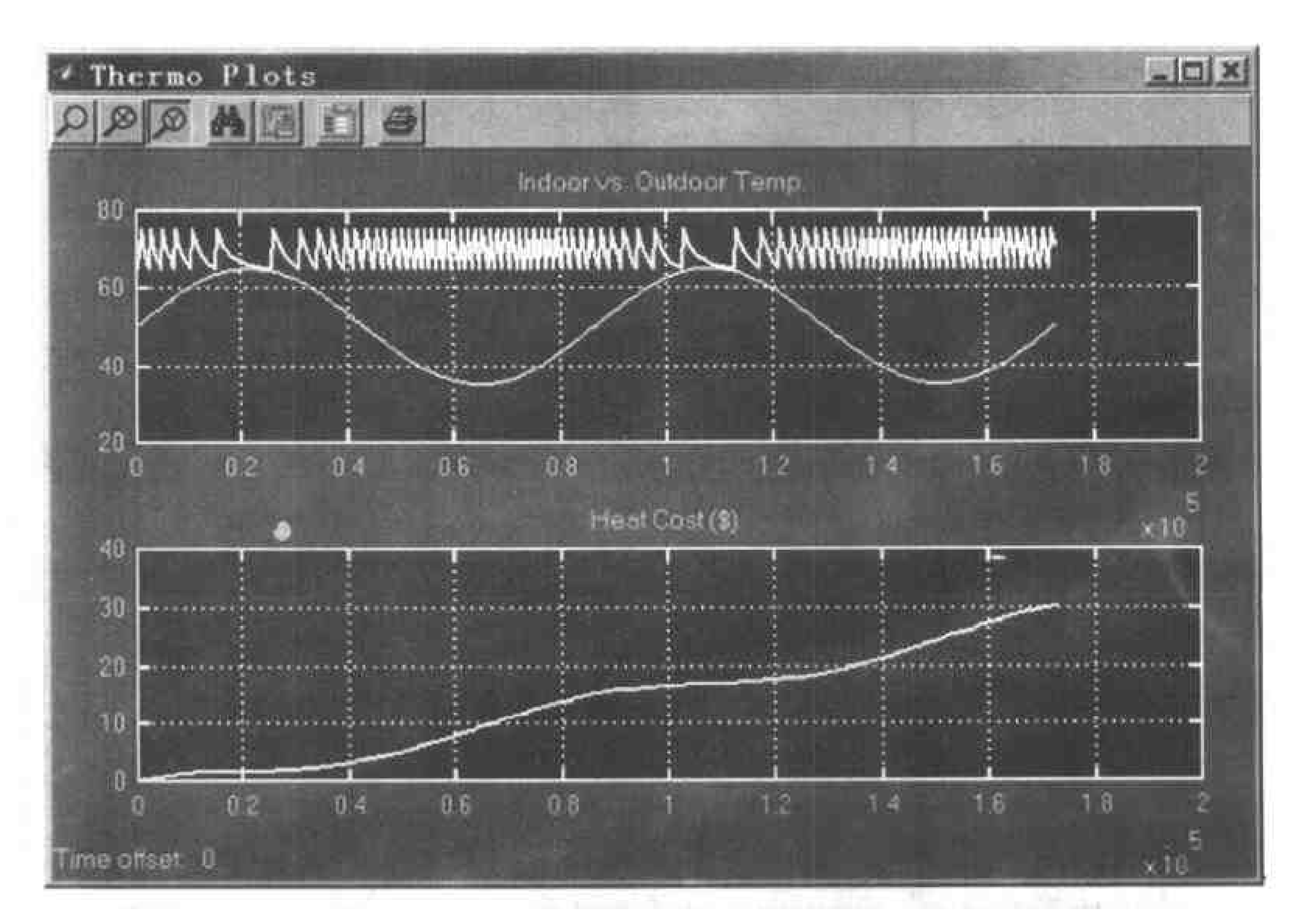


图 1.2　Scope 显示窗口

要停止仿真，从主菜单的“Simulation”下选择“Stop”菜单项，或按工具条中的“停止按钮”.

当仿真的运行停止以后，可以选择主菜单“File”下的“Close”菜单项关闭该模型.

1.2.2 示例的说明

这一示例通过一个简单的模型模拟了一个房子的热力学特性. 温度调节器的温度被设置为华氏 70 度，它受到外部温度的影响，在华氏 50 度的上下以正弦波变化，变化的幅度为 15 度. 这是为了模拟每天的温度波动.

这一模型使用了子系统来简化模型图，并且创建了可重复使用的系统. 子系统是代表子系统模块的一组模块. 这一模型包括五个子系统：一个称为温度调节器(thermostat)，一个称为房子(house)，另外三个是温度转换(temp comvert)子系统(两个将华氏温度转换为摄氏温度，一个将摄氏温度转换为华氏温度).

内部和外部的温度传给房子子系统，以此来改变房子的内部温度. 双击房子模型以查看这一子系统下的各模块. 房子子系统如图 1.3 所示.

自动温度调节器子系统模拟一个温度调节器的工作过程，决定加热系统的关闭. 双击该模块可以看到该子系统是由哪些模块组成. 自动温度调节器子系统如图 1.4 所示.

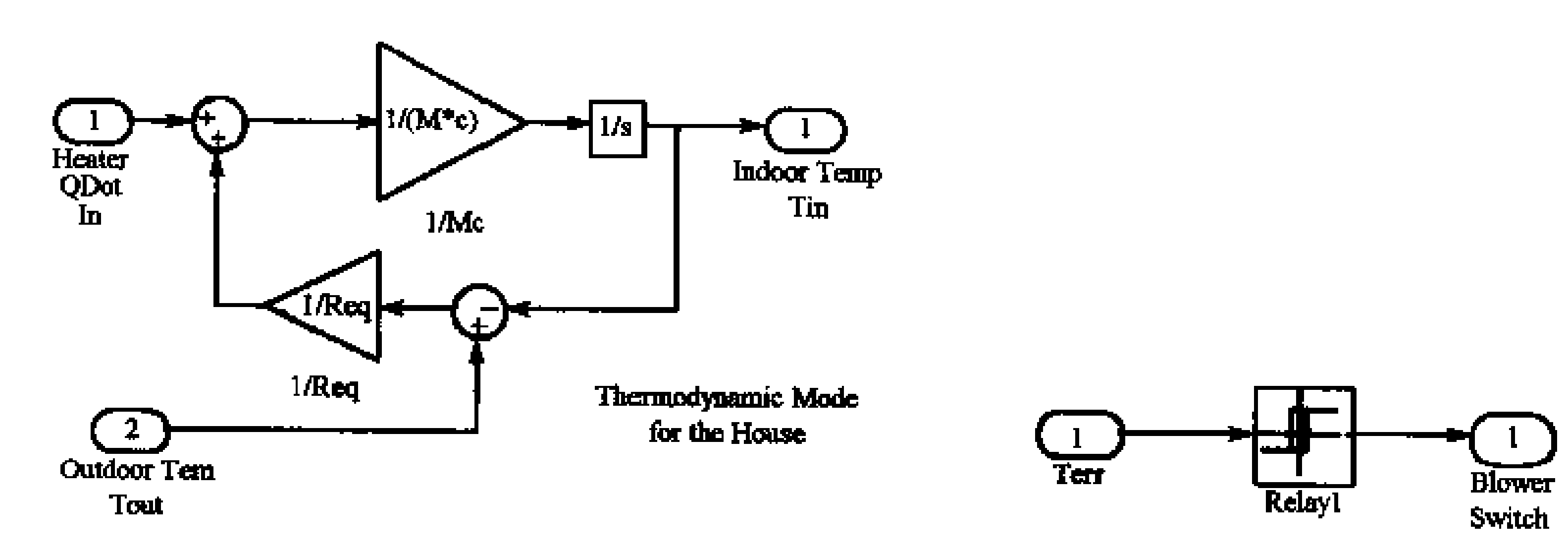


图 1.3 房子子系统 图 1.4 自动温度调节器子系统

外部和内部的温度都由同一子系统将华氏温度转换为摄氏温度. 华氏温度到摄氏温度的转换如图 1.5 所示.

当加热时，热力消费被计算出来并且显示在 Scope 模块的“Heat Cost($)”曲线图中. 内部温度被显示在“Indoor Temp”曲线图中.

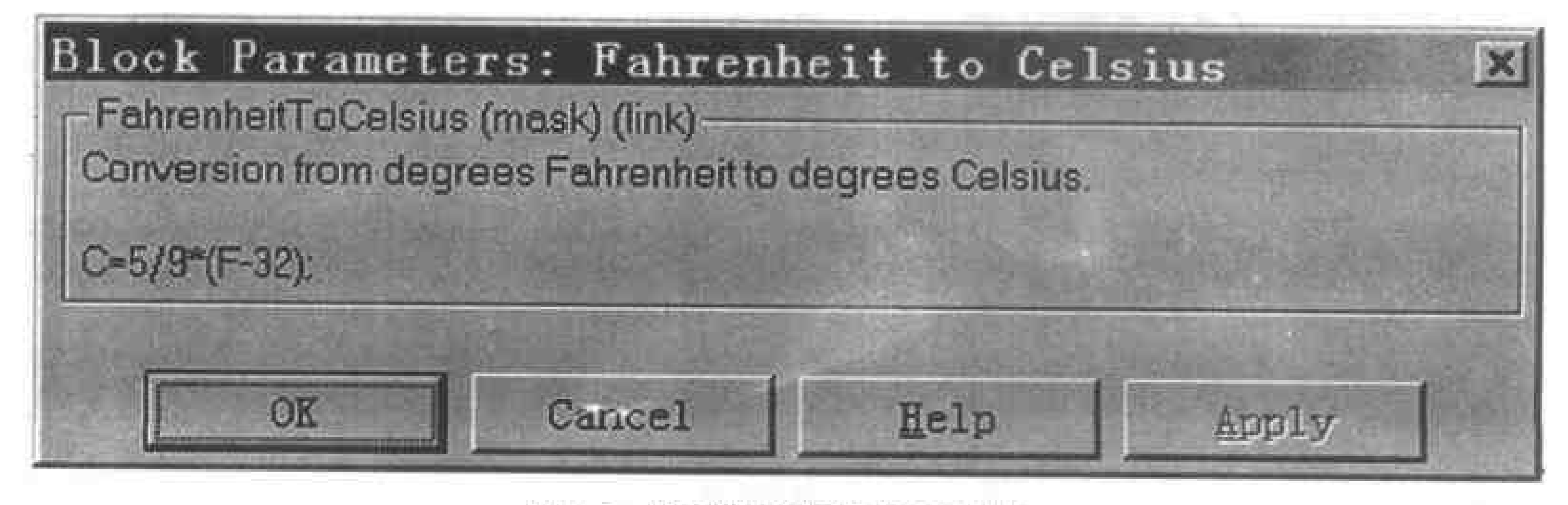


图 1.5 华氏温度到摄氏温度的转换

对模型做一些修改，看看模型对不同的参数会有些什么反应.

每一个 Scope 模块包含一个或多个信号显示区域和一些控件，这些控件使用户可以选择信号显示的范围，对信号的某一部分进行放大，或者执行其它一些有用的操作.横轴表示时间，纵轴表示信号的数值.

被标记为“Set Point”的常量(Constant)模块(在模型的左上部分)用来设置期望的内部温度.当运行仿真时，打开这一模块(双击该模块即可打开它)将温度设为 80 华氏度，看看室内温度和加热的花费是如何变化的.同样，调整室外温度(“Avg Outdoor Temp”模块)，看看它对仿真有什么影响.

通过打开标有“Daily Temp Variation”的正弦波模块并且改变其 Amplitude 参数来调整每天的温度变化.

1.2.3 建立模型的一般步骤

通过上述示例说明了建立模型时一些通常的步骤：

1) 运行仿真，包括指定参数和选择主菜单“Simulation”下的“Start”菜单项启动仿真.

2) 可以将一组相关的模块封装在一个模块内，称之为子系统.

3) 可以通过模板(masking)功能为一个模块定制一个图标(icon)，并且可以为其设计一个对话框.在“thermo”模型中，所有的子系统模块都有一个用 masking 功能定制的图标.

4) Scope 模块就像实际的示波器那样显示输出图形.Scope 模块也可显示输入信号.

1.2.4 其它一些有用的示例

另外一些示例说明了有关仿真的一些有用的概念.可以从 Simulink 的模块库窗口得到这些示例.

1) 在 MATLAB 的命令窗口中输入 Simulink3 命令.Simulink 的模块库如图 1.6 所示.

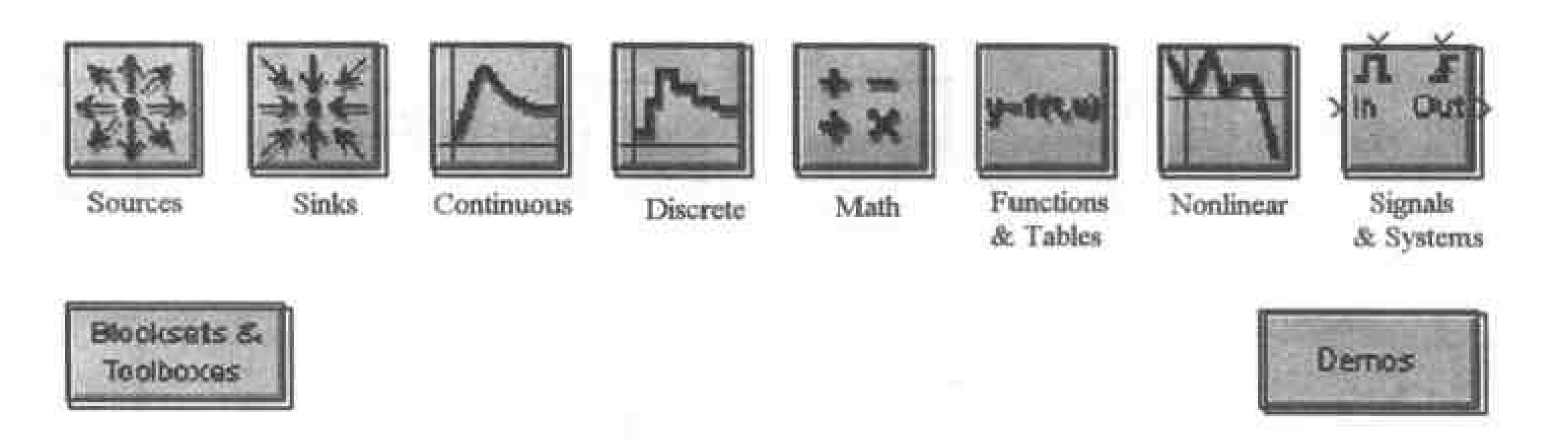


图 1.6 Simulink 的模块库窗口

2) 双击“Demos”图标.出现 MATLAB 的示例窗口.该窗口中包含了几个非常有趣的示例模型，它们展示了 Simulink 的一些有用的功能.

1.2.5 创建一个简单的模型

下面通过一个例子说明如何用建立模型的命令和操作创建模型，用户可以通过这些

命令和操作创建自己的模型.这是个对一个正弦波求积分的模型,并且显示其结果和正弦波.这一模型的模块图如图 1.7 所示.

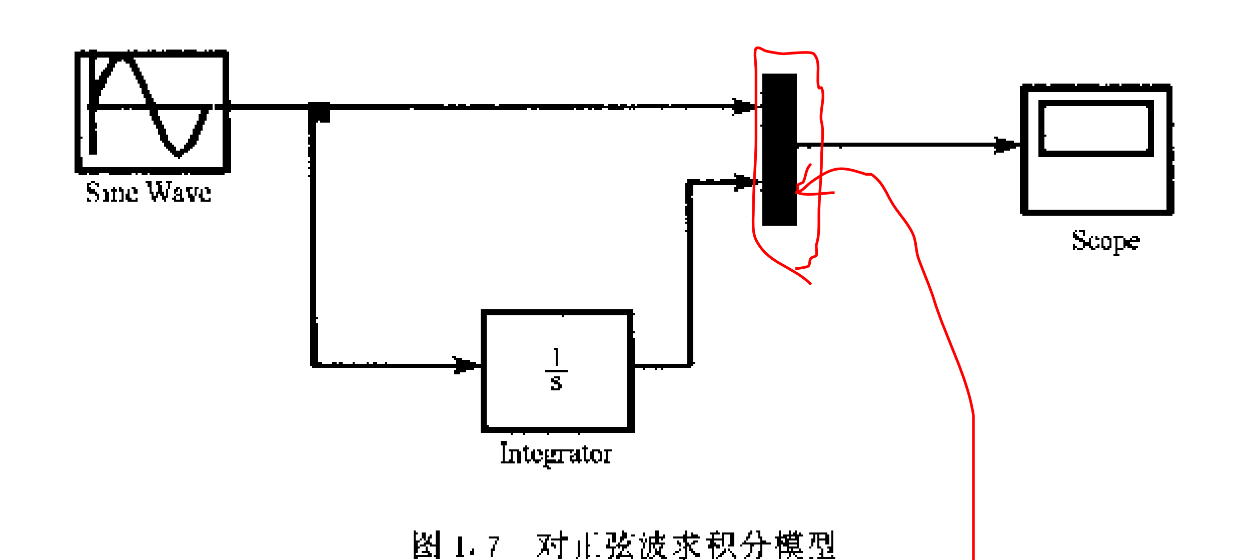


图 1.7 对正弦波求积分模型

在 MATLAB 的命令窗口中输入 Simulink 命令,将会显示 Simulink 的模块库.如果没有显示未命名窗口,可以通过 Simulink 的模块库窗口的"File|New|Model"菜单新建一个.

将模型窗口移到一个合适的位置,以便能同时看到模型窗口内容和模块库窗口的内容.在这一模型建立,要从 Simulink 以下一些模块库中复制模块:

1) 源(Sources)库[正弦波(Sine Wave)模块]
2) 接收(Sinks)库[显示(Scope)模块]
3) 线性(Linear)库[积分(Integrator)模块]
4) 连接(Connections)库[混合(Mux)模块]

打开 Sources 信号源库以取得正弦波模块.要打开模块库,只用在库的图标上双击鼠标就行了.Simulink 将会显示一个包含该库中所有模块的窗口.在 Sources 库中,所有的模块都是信号源.Sources 库的窗口显示如图 1.8 所示模块.

要想在模型中加入模块,可以先从模块库中或者从其它的模型中拷贝一个或几个模块,然后将它们粘贴到模型当中.在这里,需要拷贝的是 Sine Wave 模块.将鼠标指在 Sine

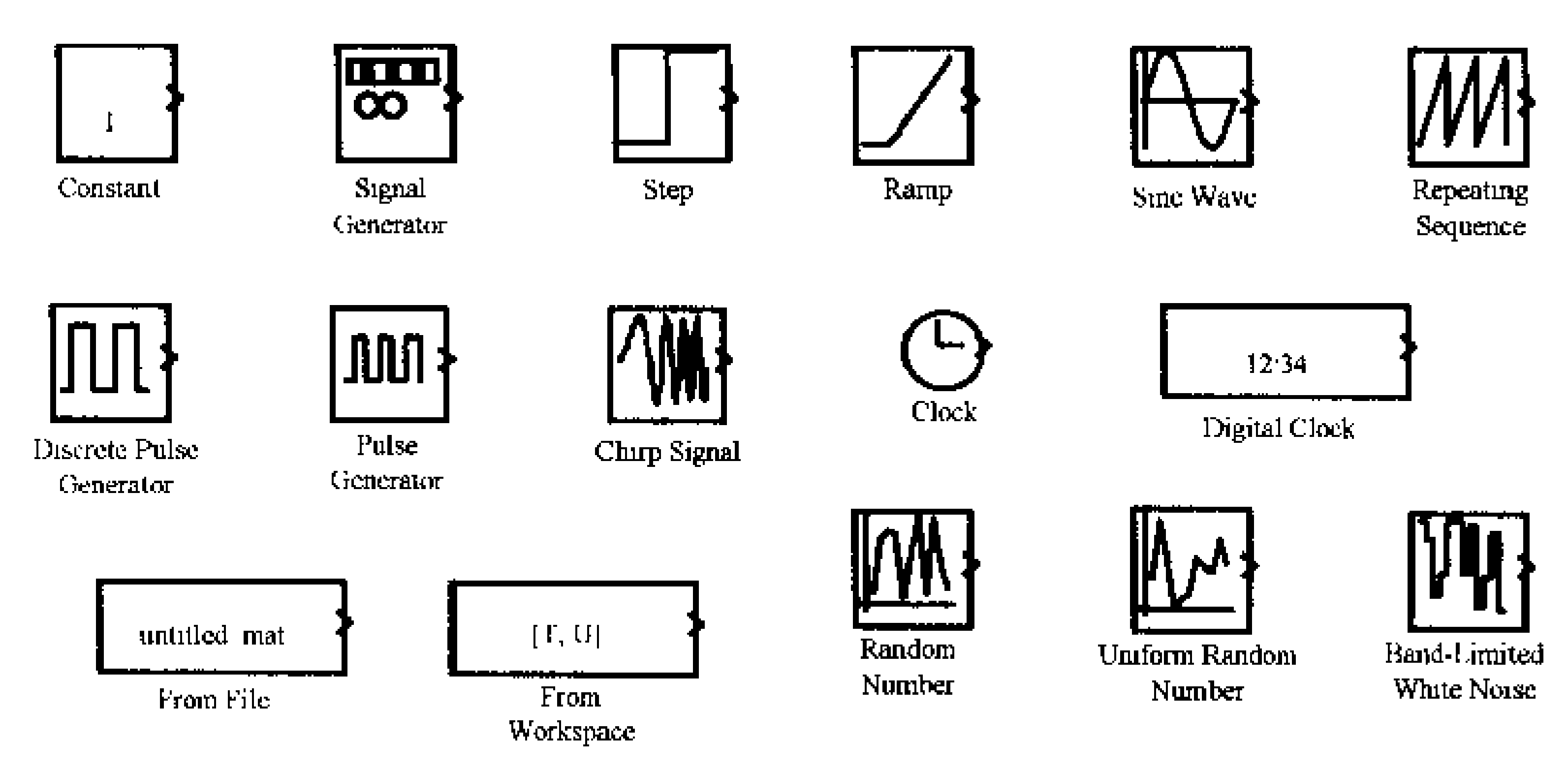


图 1.8 源(Sources)库

Wave 模块上，按下鼠标左键并保持住，当拖动模块时，会看到模块的轮廓和它的名字会随鼠标一起移动，拖动模块到模型窗口中合适的位置后放开鼠标左键. 这时，一个 Sine Wave 模块将会出现在模型窗口中.

用同样的方法将其它的模块拷贝到模型窗口. 在模型窗口中可以用与拷贝模块一样的拖拉技术移动模块在模型窗口中的位置. 还可以通过选中一个模块以后，按箭头键来对模块的位置进行微调.

当所有的模块都拷贝到了模型窗口以后，模型窗口看起来如图 1.9 所示.

如果查看一下模块的图标，将会发现在 Sine Wave 模块的右边有一个尖括号，在 Mux 模块的左边有两个尖括号. 符号"〉"，如果符号指向模块外，表示模块的输出口；而如果符号指向模块，它就是模块的输入口. 信号通过一条连接线从一个模块的输出口传到另一个模块的输入口. 当模块通过端口连接在一起时，端口的表示符号将会消失.

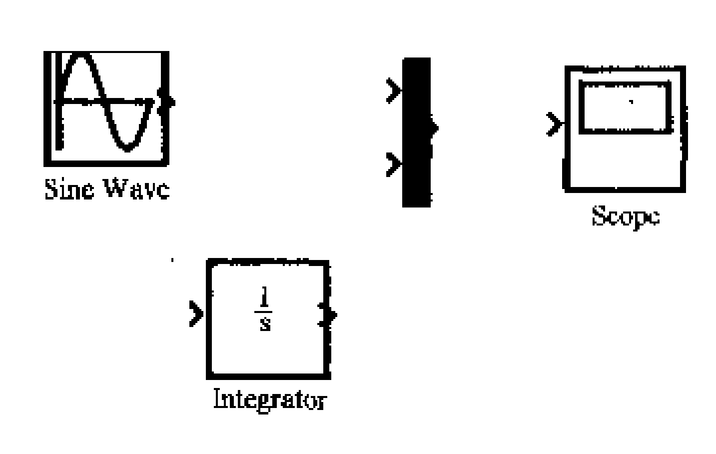


图 1.9 拖拉模块后的状态

在创建模型时将会发现 Mux 模块有两个输入端口，有时需要改变输入端口数，只要在该模块上双击鼠标以打开该模块的对话框. 将参数 Number of inputs 的值改变即可，然后单击 Close 按钮. Simulink 将会调整输入端口的个数.

现在可以将模块之间连接起来. 将 Sine Wave 模块与 Mux 模块的上部输入端口连接起来，将鼠标移到 Sine Wave 模块右边的输出端口上，这时鼠标的形状将变为"十"字形. 按下鼠标左键并拖动光标到 Mux 模块的上部输入端口，拖动光标时，从输出端口到光标之间将显示一条虚线，当光标到达 Mux 模块的输入端口时，十字光标将变为双线，这时松开鼠标按键. 两个模块之间将以一条带箭头的直线(也可能是折线)连接起来. 也可以将光标移到 Mux 模块的内部，然后松开鼠标左键，这时被连接的端口是与光标位置最接近的那个端口.

回过头再看看图 1.7 所示的模型，将会发现大部分线都是从模块的输出端到另一个模块的输入端，然而有一条线例外，它连接一条连线到一个模块的输入端口，这条线称作支线. 它将 Sine Wave 模块的输出与 Integrator 模块的输入连接起来. 要连这条线，可先将鼠标光标移到 Integrator 模块的输入端口，然后按下鼠标左键并拖动光标到连接 Sine Wave 模块与 Mux 模块的连线上放开；也可以将鼠标移动到连接 Sine Wave 模块与 Mux 模块的连线上，按下鼠标右键并拖动光标至 Integrator 模块的输入端口；或者按以下的步骤进行：

首先将光标移到连接 Sine Wave 模块与 Mux 模块的连线上.

如果是 Windows 或是 X Windows 系统，按下 Ctrl 键，如果是 Macintosh 系统，按下 Option 键，按下鼠标左键并拖动光标到 Integrator 模块的输入端口或者 Integrator 模块的图标上.

松开鼠标. Simulink 将从起点到 Integrator 模块的输入端口之间画一条线.

现在，打开 Scope 模块以观察仿真的结果. 使 Scope 窗口保持打开状态，运行仿真 10 秒钟. 首先选择菜单"Simulation"的"Parameters"以打开 Simulation parameters 窗口进行

参数设置. 在如图 1.10 所示的对话窗口中，注意 Stop time 被设为 10.0(缺省设置).

单击 Close 按钮以关闭 Simulation Parameters 对话框. Simulink 将会启用新设置的参数.

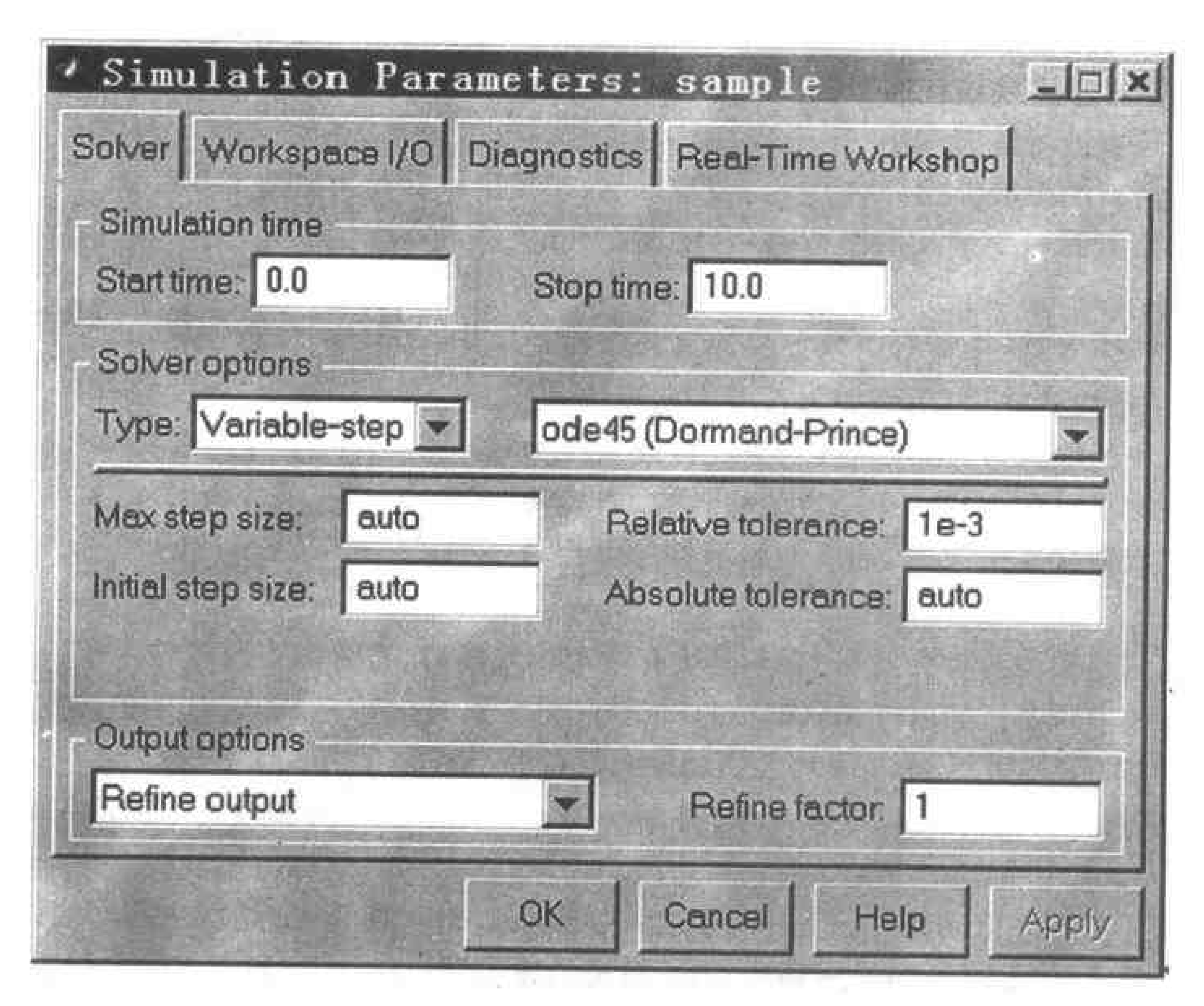


图 1.10　仿真参数设置对话窗口

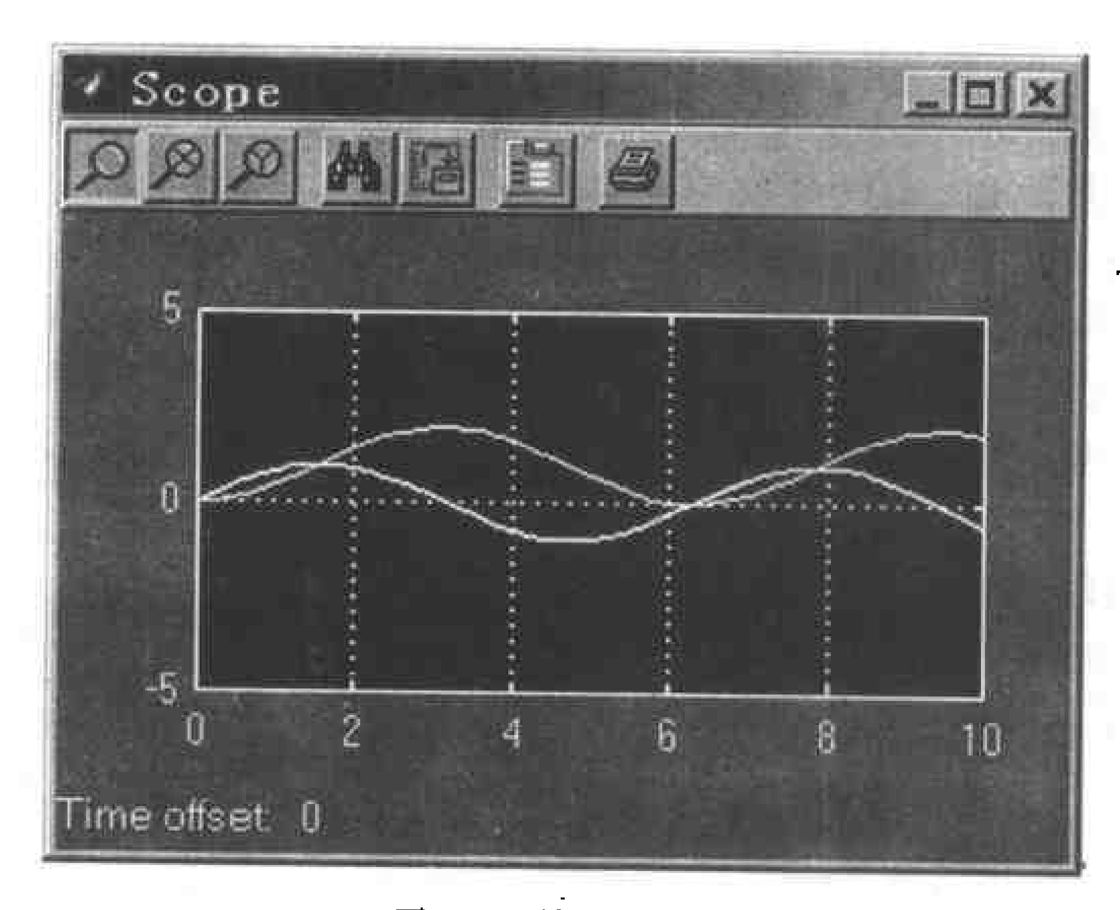


图 1.11　结果输出窗口

选择"Simulation | Start"菜单并且观察 Scope 模块的输出.结果输出窗口如图 1.11 所示.

当达到 Simulation Parameters 对话框设置的停止时间或者选择"Simulation | Stop"菜单时,仿真停止执行.

要保存该模型,选择"File | Save"菜单,并且在保存对话框中输入文件名和路径,该文件将包含模型的描述.

1.3 Simulink 是如何工作的

Simulink 模型中的每个模块都有如下特征:向量输入 u,向量输出 y 和向量状态 x.

状态向量可能由连续状态、离散状态或者它们两者的混合所组成.这些量的数学关系由下列方程所表达:

$$y = f_o(t,x,u) \qquad \text{输出}$$

$$x_{d_{k+1}} = f_u(t,x,u) \qquad \text{更新}$$

$$x_c' = f_d(t,x,u) \qquad \text{导数}$$

式中 $x = \begin{bmatrix} x_c \\ x_{d_k} \end{bmatrix}$.

仿真分初始化和仿真两个阶段.在初始化阶段:

1) 模块的参数传给 MATLAB 以求出它们的值.求出的数值结果用作实际的模块参数.

2) 去掉模型中的层次结构.每一个不是条件执行的子系统都由子系统中包含的各个模块所代替.

3) 各个模块按照它们将被更新的顺序重新排序.排序算法构造一个列表以使具有直接馈通的模块在驱动它们的模块没有被更新之前不被更新.代数循环的检测在该步进行.

4) 检查模块之间的连接以确保每一模块的输出向量的宽度与它驱动的模块输入的宽度是相同的.

初始化之后,就做好了运行仿真的准备.模型使用数值积分的方法进行仿真.提供的每一个 ODE 求解器(仿真方法)取决于模型提供其连续状态的导数的能力.计算这些导数的过程分两步.第一步,按照排序时确定的顺序计算每一个模块的输出.然后,在第二步,每一模块基于当前时间、模块的输入和状态计算其导数,得到的导数向量返回给求解器,求解器在下一时间点使用这些导数向量计算新的状态向量,新的状态向量计算出来以后,各采样数据模块和 Scope 模块也被更新.

1.3.1 过零点

Simulink 使用过零点技术检测连续信号的间断点.过零点在状态事件的处理和非连续信号的精确积分两种情况下起着重要作用.

1.3.1.1 状态事件的处理

当状态值的改变使得系统有明显的改变时系统就经历一个状态事件.状态事件的一

个简单例子是碰到地板而反弹的球.当使用变步长求解器仿真这样一个系统时,通常求解器不是采用球正好接触地面的相对应时间步.这样,球就好像越过了接触点,似乎球会穿透地面.

Simulink 使用过零点检测以确保时间步正好(在机器精度范围内)出现在时间状态事件发生的时候.因为时间步正好发生在接触地面的一瞬间,仿真就不会越过接触点并且速度从负到正的转换非常尖锐(也就是说,在不连续点处没有圆角).要看反弹球的演示,在 MATLAB 的命令窗口中输入 bounce.

1.3.1.2 非连续信号的积分

数值积分程序是基于这样的假设得到的公式,被积分的信号是连续的并且有连续的导数.如果在某一仿真步遇到了不连续的情况(状态事件),Simulink 使用过零点检测以找到不连续是在何时发生的,然后该积分步就积分到不连续点的左边沿,最后,Simulink 跨过不连续点,在信号的下一分段连续的部分开始新的积分步.

1.3.1.3 实现细节

Simulink 的模块中使用过零点的一个例子是 Saturation 模块.过零点检测 Saturation 模块中有如下状态事件:

1) 输入信号到达上限;

2) 输入信号离开上限;

3) 输入信号到达下限;

4) 输入信号离散下限.

Simulink 中定义其状态事件的模块,认为具有固有的过零点.如果需要过零点事件的明确说明,则可使用 Hit Crossing 模块.

状态事件的检测取决于内部的过零点信号的构造.从模块图中不能获取该信号,对于 Saturation 模块,用来检测上限过零点的信号是 zcSignal=UpperLimit-u,其中 u 是输入信号.

过零点信号具有方向属性,它可以取如下的值:

rising:当信号升到或升过 0,或者当信号离开 0 而成为正值时发生过零点.

falling:当信号降到或降过 0,或者当信号离开 0 而成为负值时发生过零点.

either:在上述两种情况下时都会发生过零点.

对于 Saturation 模块的上限,过零点的方向是 either.这就使得使用相同的过零点检测信号能够检测进入和离开饱和状态的事件.

如果误差限太大,Simulink 有可能不能检测出过零点.例如,如果过零点发生在某一时间步内,但是该步的起始值和末尾的值在符号上没有改变,求解器会跨过该过零点而不能够检测到其存在.

图 1.12 显示了一个过零点的信号.在第一种情况,积分器跨过了该事件,而在第二种情况,求解器检测到了该事件.

如果怀疑漏掉了过零点检测,可将误差限设小一些以确保求解器采用足够小的步长.

有可能创建的模型出现高频不连续的波动.这样的系统通常是物理上不可实现的.因

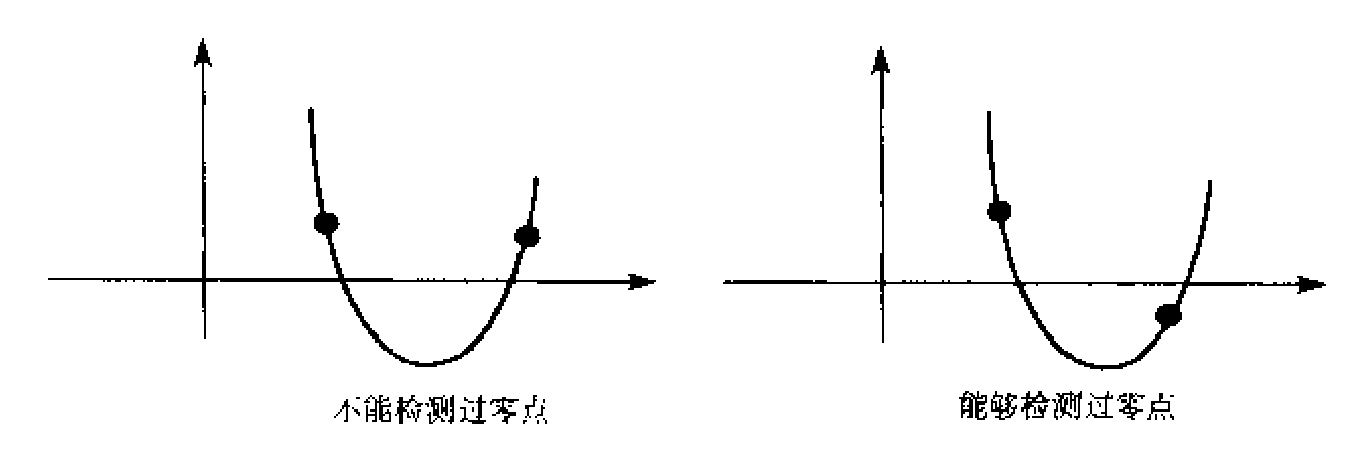


图 1.12 检测过零点的两种方法

为不连续高频出现而导致反复的过零点检测，仿真的步长变得非常小，必然导致仿真终止.

如果怀疑模型出现了这种情况，可以通过选取 Simulation Parameters 对话框中的 Diagnostics 页中的 Disable zero crossing detection 复选框使模型不进行过零点检测. 尽管不进行零点检测可以解决这一问题，但是将不再有过零点检测所带来的精度的提高.

表 1.1 所列的这些模块具有过零点.

表 1.1 有过零点的模块

模 块	过零点描述
Abs	一个：检测输入信号什么时候以上升或者下降的方向穿过零点.
Backlash	两个：一个用来检测什么时候使用上阈值，另一个用于检测什么时候使用下阈值.
Dead Zone	两个：一个用来检测什么时候进入死区(输入信号减去下限)，另一个用来检测什么时候脱离死区(输入信号减去上限).
Hit Crossing	一个：检测什么时候输入穿过阈值. 这一过零点不受 Simulation Parameters 对话框中 Disable zero crossing detection 复选框的影响.
Integrator	如果存在复位端口，检测什么时候复位发生. 如果输出受限制，有三个过零点：一个检测什么时候达到上饱和限，一个检测什么时候达到下饱和限，另一个检测什么时候脱离饱和.
MinMax	一个：对于输出向量的每一个元素，检测输入信号什么时候是新的最小值或者最大值.
Relay	一个：如果继电器处于关的状态，检测打开的时刻. 如果继电器处于开的状态，检测关掉的时刻.
Relational Operator	一个：检测什么时候输出改变.
Saturation	两个：一个检测什么时候达到或者离开上限，另一个检测什么时候达到或者离开下限.
Sign	一个：检测输入什么时候穿过 0.
Step	一个：检测阶跃的时间.
Subsystem	对于条件执行的子系统：一个提供给激活端口(如果存在)，一个提供给触发端口(如果存在).
Switch	一个：检测什么时候转换条件发生.

1.3.2 代数回路

Simulink 的一些模块具有直接馈通的输入端口.这意味着在不知道从这些输入端口进入模块的信号值的情况下,就不能计算出这些模块的输出.具有直接馈通的输入端口的模块的例子有:

1) Elementary Math 模块;

2) Gain 模块;

3) Integrator 模块的重置端口和初始状态端口;

4) Product 模块;

5) State-Space 模块(当有非零 D 矩阵时);

6) Sum 模块;

7) Transfer Fcn 模块(当分子和分母具有相同的阶数时);

8) Zero-Pole 模块(当零点数与极点数相等时).

当具有直接馈通的输入端口被同一模块的输入直接驱动,或经过由另外的具有直接馈通的模块构成的反馈回路驱动时,就会出现代数回路.一个最简单的例子是如图 1.13 所示的标量回路.

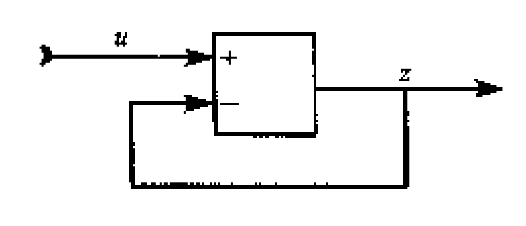


图 1.13　简单标量回路

从数学上讲,这一回路表明 Sum 模块的输出是状态 z 约束,等于第一个输入 u 减去代数状态 z(即 $z=u-z$).这一简单回路的结果是 $z=u/2$,但是多数代数回路不能够通过观察得到结果.很容易用多个代数状态变量 $z1$,$z2$,等等,创建向量代数回路,如图 1.14 所示的模型.

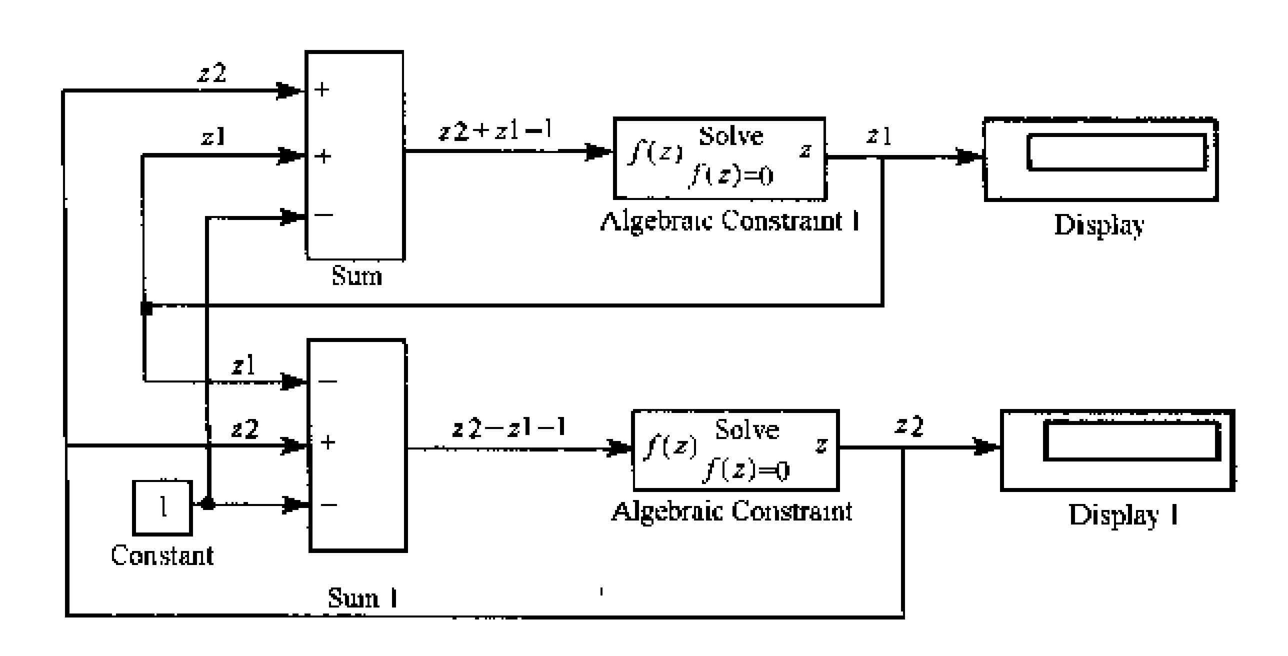


图 1.14　向量回路模型示例

该 Nonlinear 库中的代数约束(Algebraic Constraint)模块是模拟代数方程并指定初始估计值的一种简便方法.该 Algebraic Constraint 模块约束它的输入信号 $F(z)$ 为 0,并且输出一个代数状态 z.输出必须通过一些反馈环节影响输入.可以在模块的对话框中提供代数状态值的初始估计值以改进代数回路求解器的效率.

标量代数回路代表标量代数方程或者 $f(z)=0$ 形式的约束,其中 z 是回路中的一个模块的输出,函数 f 由经过回路中另外的模块到该模块输入的反馈环节所组成.在前面

仅有一个模块的例子中，$f(z)=z-(u-z)$. 在前面给出的向量回路例子中，方程如下：

$$\begin{aligned} z_2 + z_1 - 1 &= 0 \\ z_2 - z_1 - 1 &= 0 \end{aligned} \tag{1.1}$$

当模型中包含有代数约束 $f(z)=0$ 时，会出现代数回路. 当要建模的系统存在物理上的互相连通性时，会出现这样的约束；特别地，当试图模拟微分/代数系统(DAE)时也可能出现这样的约束.

当模型包含有代数回路时，Simulink 在每一时间步都调用回路求解程序. 回路求解器迭代执行以确定问题的计算结果(如果可能). 这样，有代数回路的模型运行起来比没有代数回路的模型要慢.

要计算 $f(z)=0$，Simulink 的回路求解器使用牛顿法. 尽管该方法是稳定的，但是，在代数状态 z 的初始估计值设定不恰当的情况下，可能产生使回路求解器不收敛的回路. 可以在代数回路的一条信号线上，放置一个 IC 模块(通常用来指定信号的初始状态)，以指定它的初始估计值. 另外一种指定代数回路中信号线的初始估计值的方法是使用代数约束(Algebraic Constraint)模块，如图 1.14 所示.

只要有可能，要用 IC 模块或者 Algebraic Constraint 模块指定回路中代数状态变量的初始估计值.

1.3.3 非代数直接馈通回路

与所有直接馈通回路是代数的一般情况不同，也有一些例外的情况.

1) 回路包含触发子系统；

2) 回路从输出连接积分器的复位端口.

在触发子系统中，求解器可以安全地假设子系统的输入在触发时是稳定的. 这就允许使用前一时间步的输出来计算当时间前步的输入，因此排除了代数回路求解器的需要，如图 1.15 所示的系统.

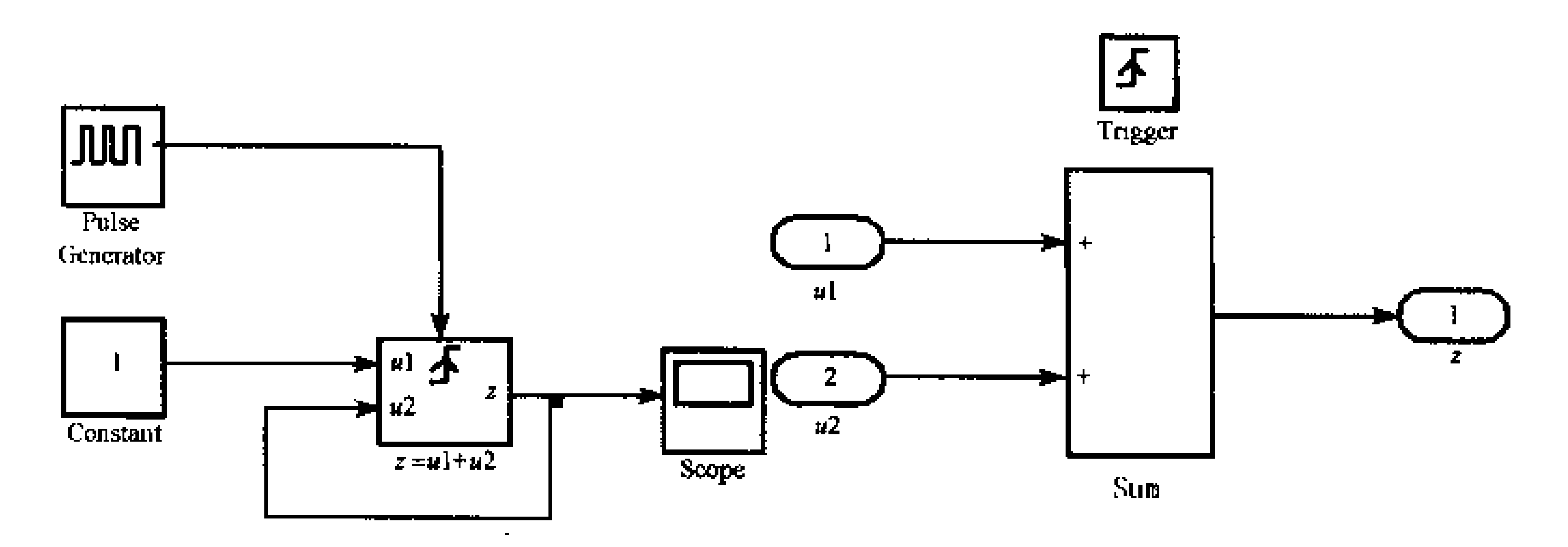


图 1.15 非代数直接馈通回路示例

该系统可以有效地求解方程式：$z = 1 + u$；其中 u 是子系统触发最后的 z 值. 系统的输出在系统显示器上显示为阶梯函数，如图 1.16 所示. 那么，在该示例中如果去除系统中的触发器，如图 1.17 所示，结果又会怎样呢？此时，每个时间步加法器子系统 $u2$ 端口的输入等于当前步子系统的输出. 而系统的数学表达式为：$z = z + 1$，表明没有合法数学

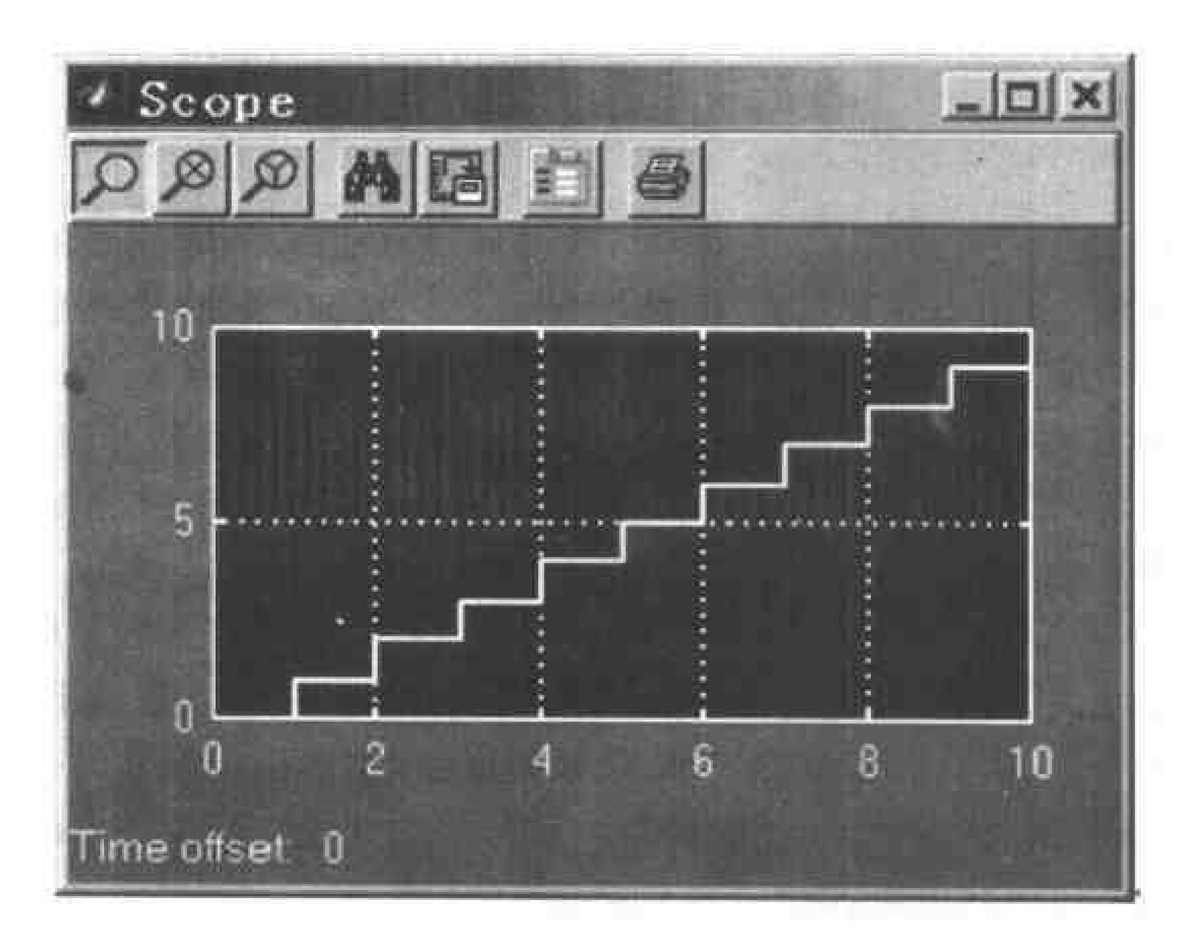


图 1.16　示例模型运行结果

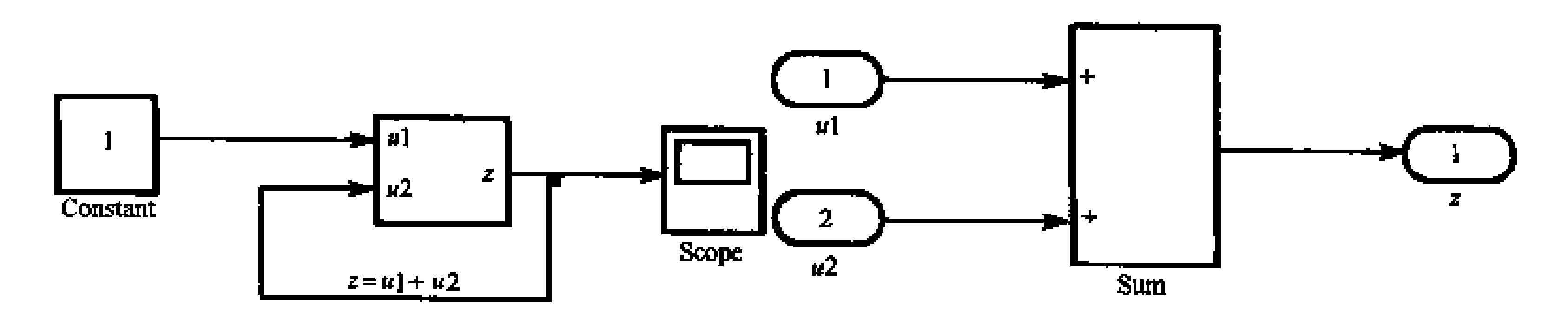


图 1.17　将图 1.15 模型中的触发器去除

结论.

1.3.4　不变的常量

模块要么具有明确指定的采样时间,或者从馈送它们的模块(或由它们馈送的模块)那儿继承采样时间.

Simulink 指定 Constant 模块的采样时间是无穷大,也被称作常数采样时间(constant sample time). 如果另外的模块从 Constant 模块那里接收输入,它们将具有常数采样时间,而不从另外的模块那里继承采样时间. 这就意味着这些模块的输出在仿真期间不会改变,除非模型的用户明确地改动它们.

例如,在如图 1.18 所示的模型中,Constant 模块和 Gain 模块都具有常数采样时间.

因为 Simulink 支持在仿真期间改变模块参数的能力,所有模块甚至包括具有常数采样时间的模块,在模型的有效采样时间点都必须生成它们的输出.

因为这一特性,所有的模块在每一采样时间点都计算它们的输出,或者,如果是纯连续系统,在每一仿真步都计算它们的输出. 对于在仿真期间参数不会改变的具有常数采样时间的模块,在仿真期间计算这些模块会减慢仿真的速度.

可以设定不变常量(InvariantConstants)参数,以从仿真回路中除去所有具有常数采

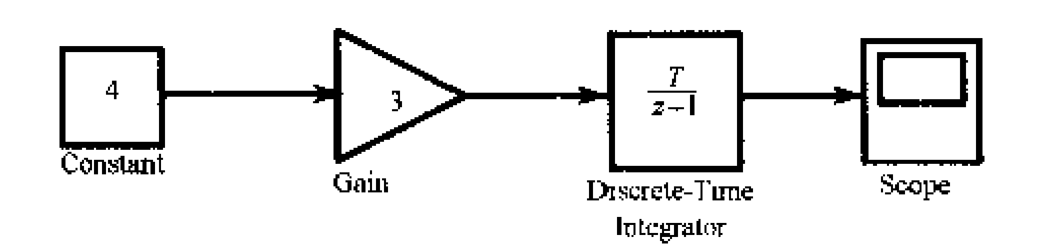


图 1.18 具有常数采样时间的模型

样时间的模块.这一特性的作用是双重的:第一,这些模块的参数在仿真期间不能够被改动;第二,仿真速度会得到提高.速度提高的程度取决于模型的复杂程度、具有常数采样时间的模块的数目和仿真的有效采样速率.

可以通过输入下面的命令设定模型的参数:

set_param('model_name', 'InvariantConstants', 'on')

可以用如下的命令关掉这一特性:

set_param('model_name', 'InvariantConstants', 'off')

可以通过选择 Format 菜单下的 Sample Time Colors 菜单项,确定哪些模块具有常数采样时间.具有常数采样时间的模块用一种特定颜色来显示.

1.3.5 离散时间系统

Simulink 具有仿真离散(采样数据)系统的能力.模型可以是多采样率的,也就是说,它们可以包含有以不同的速率采样的模块.模型还可以是既包含有离散模块,又包含有连续模块的混合模型.

(1) 离散模块

每一个离散模块在其输入都有一个内置的采样器,在其输出有一个 0 阶控制.当离散模块与连续模块混合在一起时,在采样时间之间,离散模块的输出保持为常量.离散模块的输出只在采样时间点上被更新.

(2) 采样时间

采样时间(Sample time)参数设定离散模块状态改变的采样时间.通常,采样时间被设成标量变量;然而,它也可能通过在该参数域中指定一个包含有两个元素的向量来指定一个时间偏移量.

例如,指定 Sample time 参数为向量[Ts, offset]即是设定采样时间为 Ts,设定偏移量为 offset.该离散模块只在采样时间的整数倍加上偏移量处改变:t=n * Ts+offset;式中 n 是整数,offset 可以是正数也可以是负数,但比采样时间小.如果一些离散模块必须比另外一些离散模块更新得早一些或晚一些时,偏移量就是很有用的.

当仿真正在运行时,不能改变模块的采样时间.如果需要改变模块的采样时间,必须停止并重启动仿真以使改变生效.

(3) 纯离散系统

纯离散系统能够用任何求解器进行仿真,而得到的结果没有区别.选择任一种离散求解器,只在采样点上产生输出点.

(4) 多采样率系统

多采样率系统中模块的采样速率不同.这些系统可以全用离散模块,或者既用离散模

块又用连续模块进行建模. 如图 1.19 所示的是一简单多采样率离散模型.

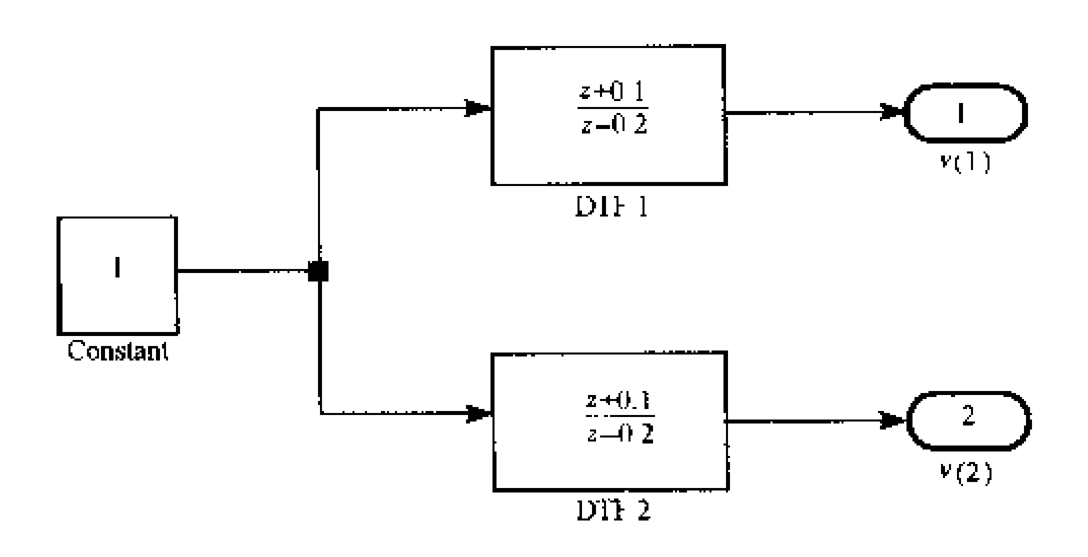


图 1.19 多采样率离散模型示例

对于该例, DTF1(Discrete Transfer Fcn 模块)的采样时间被设为[1 0.1], 即其偏移量为 0.1; DTF2(Discrete Transfer Fcn 模块)的采样时间被设为 0.7, 无偏移. 将该模型命名为"sample5.mdl"保存, 运行仿真, 并使用 stairs 函数画出输出图形.

```
[t, x, y]=sim('sample5', 3);
stairs(t, y)
```

生成的图形如图 1.20 所示.

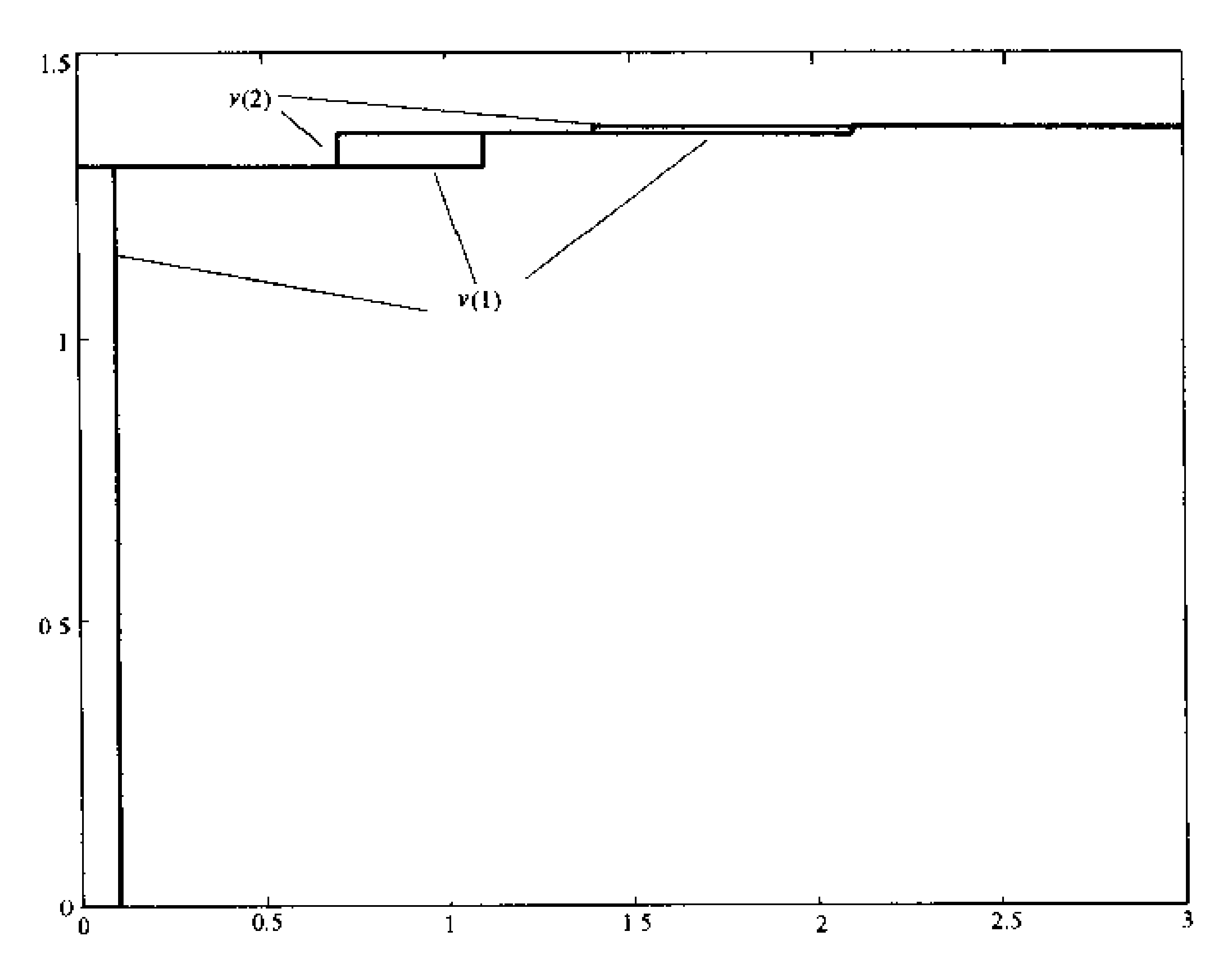


图 1.20 多采样率离散模型示例运行结果

对于 DTF1 模块, 它的偏移量是 0.1, 所以直到 $t=0.1$ 它才有输出. 因为传递函数的初始条件是 0, 所以在 $t=0.1$ 之前 DTF1 的输出 $y(1)$ 是 0.

(5) 采样时间颜色

Simulink 使用采样时间颜色特性识别模型中不同的采样率，该特性通过表 1.2 给出的颜色方案显示不同的采样率.

表 1.2 采样时间颜色

颜 色	用 途
黑色	连续模块
洋红色	常数模块
黄色	混合(多个子系统组成的模块，或者 Mux 模块或者 Demux 模块组合具有的不同采样时间的信号)
红色	最快的离散采样时间
绿色	第二快的离散采样时间
蓝色	第三快的离散采样时间
浅蓝色	第四快的离散采样时间
深绿色	第五快的离散采样时间
青色	被触发的采样时间
灰色	最小时间步是固定的

要理解这一特性是如何工作的，熟悉 Simulink 的采样时间传播引擎(STPE)是很重要的. 图 1.21 说明了一个采样时间为 Ts 的 Discrete Filter 模块，驱动一个 Gain 模块. 因为 Gain 模块的输出只是其输入乘以一个常数，它的输出改变的速率与滤波器相同. 也就是说，Gain 模块的采样速率等于滤波器的采样速率. 这就是 STPE 的基本机制.

要启用采样时间颜色特性，选择 Format 菜单下的 Sample Time Colors 菜单项. 在每次对模型进行改动时，Simulink 并不自动更新模型的颜色，因此必须选择Edit 菜单下的 Update Diagram 菜单项以更新模型的颜色. 要返回到原来的颜色，再一次选择 Sample Time Colors.

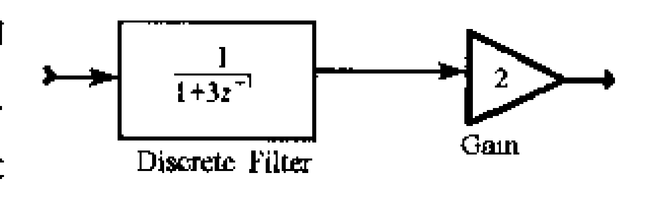


图 1.21 STPE 的示例

如果使用采样时间颜色，分配给各个模块的颜色，取决于模型中其采样时间与其它模块的采样时间的比较.

Simulink 根据下面这些原则为单个模块设置采样时间：

1) 连续模块(如 Integrator，Derivative，Transfer Fcn 等等)是连续的.

2) 常数模块(例如 Constant)是常数的.

3) 离散模块(例如 Zero-Order Hold，Unit Delay，Discrete Transfer Fcn 等等)具有用户在模块对话框中明确指定的采样时间.

4) 所有其它的模块，具有基于其输入采样时间的隐含指定的采样时间. 例如，跟在 Integrator 模块后面的 Gain 模块被看作是连续模块；而跟在 Zero-Order Hold 模块后面的 Gain 模块被看作是离散模块，且它的采样时间与它前面的 Zero-Order Hold 模块相同.

对于那些各个输入具有不同采样时间的模块，如果所有采样时间是最快采样时间的整数倍，模块将被赋予最快输入的采样时间. 如果使用了变步长求解器，模块将被赋予连续采样时间. 如果使用的是定步长求解器，并且采样时间的最大公约数(基本采样时间)能够被计算出来，就把它作为模块的采样时间，否则使用连续采样时间.

注意到 Mux 和 Demux 模块只是简单的组操作符，传过它们的信号保持其时间信息. 因此，从 Demux 模块发出的信号线（如果它们是由不同采样时间的信号源驱动的），可能具有不同的颜色. 在这种情况下，Mux 和 Demux 模块用（黄色）颜色标注为混合模块，以表明它们处理多采样率的信号.

同样，包含有不同采样时间模块的子系统（Subsystem）模块，也用黄色表示为混合模块，因为没有一种单一的采样率与它们相联系. 如果子系统中的所有模块运行于单一的采样率，这样 Subsystem 模块根据该采样率着色.

在有些情况下，如果不影响仿真的输出，Simulink 也反向传送采样时间给它的源模块. 在如图 1.22 所示的模型中，Simulink 识别出是信号发生器（Signal Generator）驱动离散时间积分器（Discrete-Time Integrator），所以它指定 Signal Generator 模块和 *Gain* 模块的采样时间与 Discrete-Time Integrator 模块的采样时间相同.

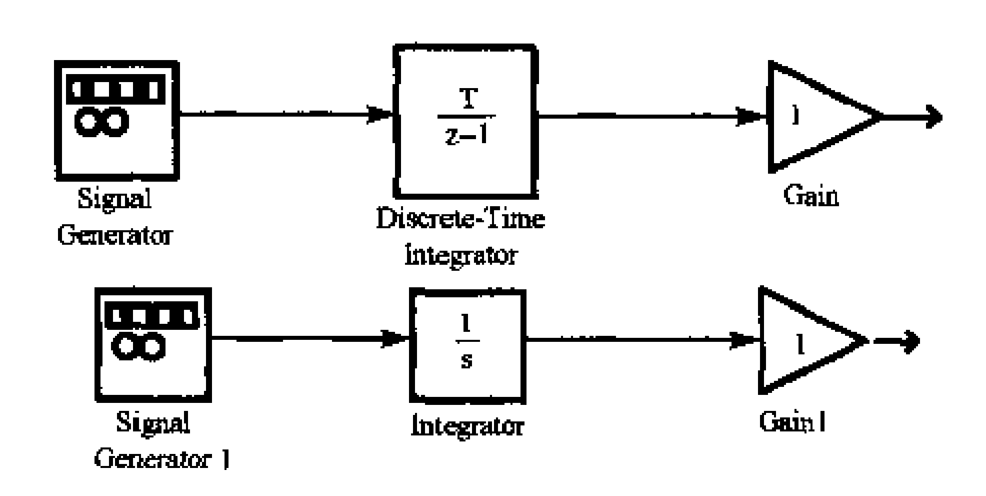


图 1.22　反向传送采样时间示例

可以通过选取 Sample Time Colors，并且注意到所有的模块都着成红色证实这一点. 因为 Discrete-Time Integrator 模块在采样时间只注视其输入，所以这一改变不影响仿真的结果，但会导致性能的改进.

用连续的 Integrator 模块代替 Discrete-Time Integrator 模块，选择 Edit 菜单下的 Update Diagram 菜单项给模型重新着色会使 Signal Generator 和 Gain 模块变成连续模块，它们全被着成黑色.

（6）混合连续和离散系统

混合的连续和离散系统由离散的模块和连续模块组成. 这样的系统能够用各种综合方法进行仿真，其中某些方法会比另外的方法更有效、更精确. 对于大多数混合的连续和离散系统，龙格-库塔（Runge-Kutta）变步长法，ode23 和 ode45 在效率和精度方面优于其它的方法. 由于不连续性与离散模块的采样和保持有关，建议在混合连续和离散系统中，不要使用 ode15s 和 ode113 这两种方法.

第二章　Simulink 模型创建

2.1　启动 Simulink

要启动 Simulink 必须先启动 MATLAB. 启动 MATLAB 后，有两种方法可以启动 Simulink：

1）在 MATLAB 的工具条中单击 Simulink 的按钮 ▇；

2）在 MATLAB 的命令窗口中输入 Simulink 命令.

在 Windows 平台上启动 MATLAB 的 Simulink 后，将会显示 Simulink 的模块库浏览(Simulink Library Browser)窗口，如图 2.1 所示. 同时，可以在库浏览窗口的 Simulink 节点上，通过点击鼠标右键，打开 Simulink 模块库窗口，如图 1.6 所示.

Simulink 的模块库浏览窗口，显示了系统中安装的一个树状结构的 Simulink 模块库. 创建模型时，将从这些模块库浏览窗口中拷贝模块到模型窗口.

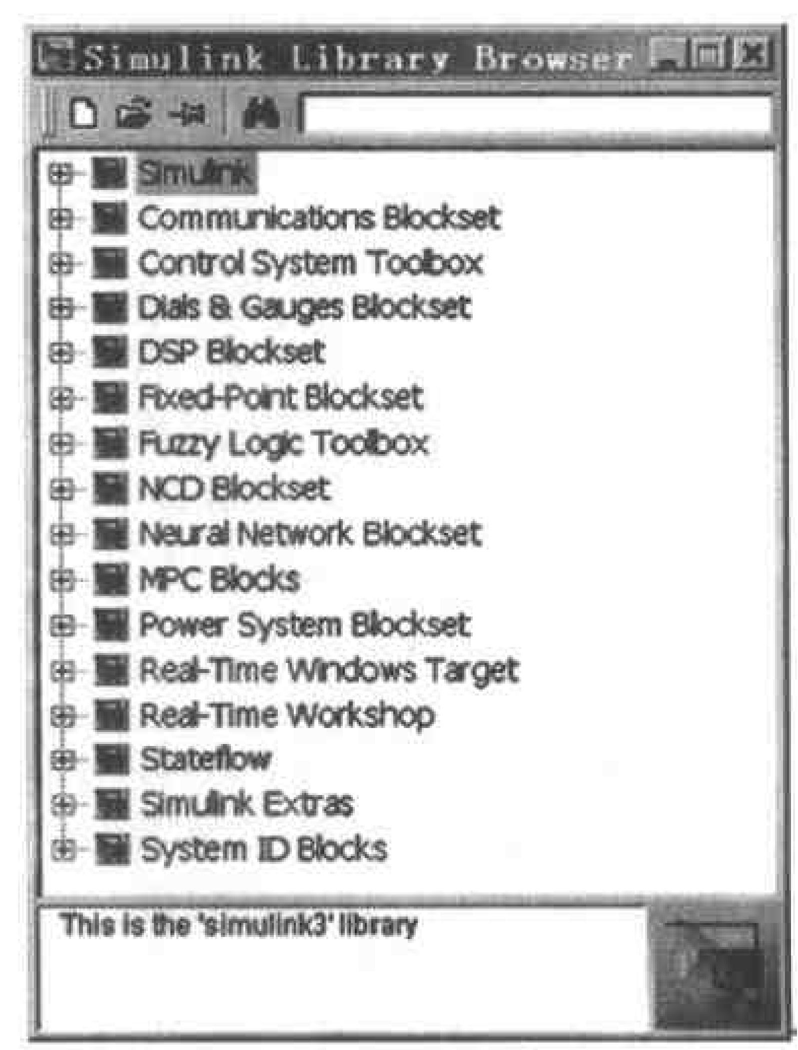


图 2.1　Simulink 的模块库浏览窗口

要运行 Simulink 和对模型进行操作，可从 Simulink 的菜单中选择菜单项，或者在 MATLAB 的命令窗口中输入命令.

2.1.1　Simulink 窗口

Simulink 使用不同的窗口分别显示模块库、模型和仿真图形输出. 这些窗口不是 MATLAB 图形(figure)窗口，因此不能使用其图形操作(handle graphics)命令进行操作.

Simulink 的所有窗口的大小，都被调节成能够适应大多数的显示器分辨率. 如果显示器的分辨率特别高或者特别低，都会发现这些窗口太小或太大. 如果出现这种情况，调整窗口尺寸，并保存模型，就可保存所做的调整.

2.1.2　创建新的模型

要创建新的模型，点击库浏览窗口的工具条中的"新建一个模型"按钮即可；或者在库窗口中，选择File 菜单下的New 子菜单，并选择 Model 菜单项. 这些窗口与其它窗口一样可以移动.

2.1.3 编辑已存在的模型

编辑一个已存在的模型图,有两种方法:

1) 在库浏览窗口中,点击工具条中的"打开一个模型"按钮;或在库窗口中,从 File 菜单下选择Open 菜单项,然后选择或者输入要编辑的模型的文件名;

2) 在 MATLAB 命令窗口中输入模型的名字(不带.mdl 扩展名).模型必须在当前的目录或者路径下.

2.1.4 输入 Simulink 命令

通过输入命令来运行 Simulink 或自己建造的模型,可以通过如下方式输入命令:

1) 从 Simulink 菜单条上选择项目.Simulink 菜单条在每个模型窗口的最上部,菜单条的命令应用于窗口中的内容.

2) 从 Simulink 前后相关的菜单中选择项目.在 Simulink 的 Windows 版本中,当在一个模型或模块库窗口上,点击鼠标右键时,就会显示前后相关的菜单.菜单的内容取决于是否选中模块.如果选中了一个模块,菜单显示仅仅适用于所选模块的命令;如果没有选择模块,菜单显示的命令作用于一个整体模型或库.

3) 按击 Simulink 工具条按钮.在 Simulink 的 Windows 版本中,模型窗口的 Simulink 菜单条之下,可以显示一个工具条.在 Simulink 的View 菜单中可控制工具条选项.工具条中包含经常使用的 Simulink 命令按钮,如打开、运行、关闭模型等.可以通过按相应按钮的办法来执行这些命令.如打开一个 Simulink 模型,就按含有打开文件图标的按钮.可以通过移动鼠标指向某一按钮,一个含有描述该按钮文字的窗口会出现,该窗口称为工具提示,由此可以了解执行哪个命令按钮.

4) 在 MATLAB 命令行窗口中输入命令.当运行仿真并分析其结果时,可以在 MATLAB 命令行窗口输入 MATLAB 命令.

2.1.5 取消和重做命令

在模型编辑窗口(见图 2.2),可以选择Edit 菜单下的 Undo 菜单项,或者按工具条中的"取消"按钮,或者用 Ctrl+Z 组合键,以取消刚刚执行的操作,可以连续取消前面多达 101 次的操作.可以取消的操作包括:

1) 添加或者删除模块;

2) 添加或者删除连线;

3) 添加或者删除模型的注释;

4) 编辑模块的名字.

在模型编辑窗口,可以选择Edit 菜单下的Redo 菜单项,或者按工具条中的"重做"按钮,或者用 Ctrl+Y 组合键,以恢复 Undo 命令所取消的操作.

2.1.6 缩放模块框图

Simulink 允许在当前的 Simulink 窗口放大或缩小模块图.

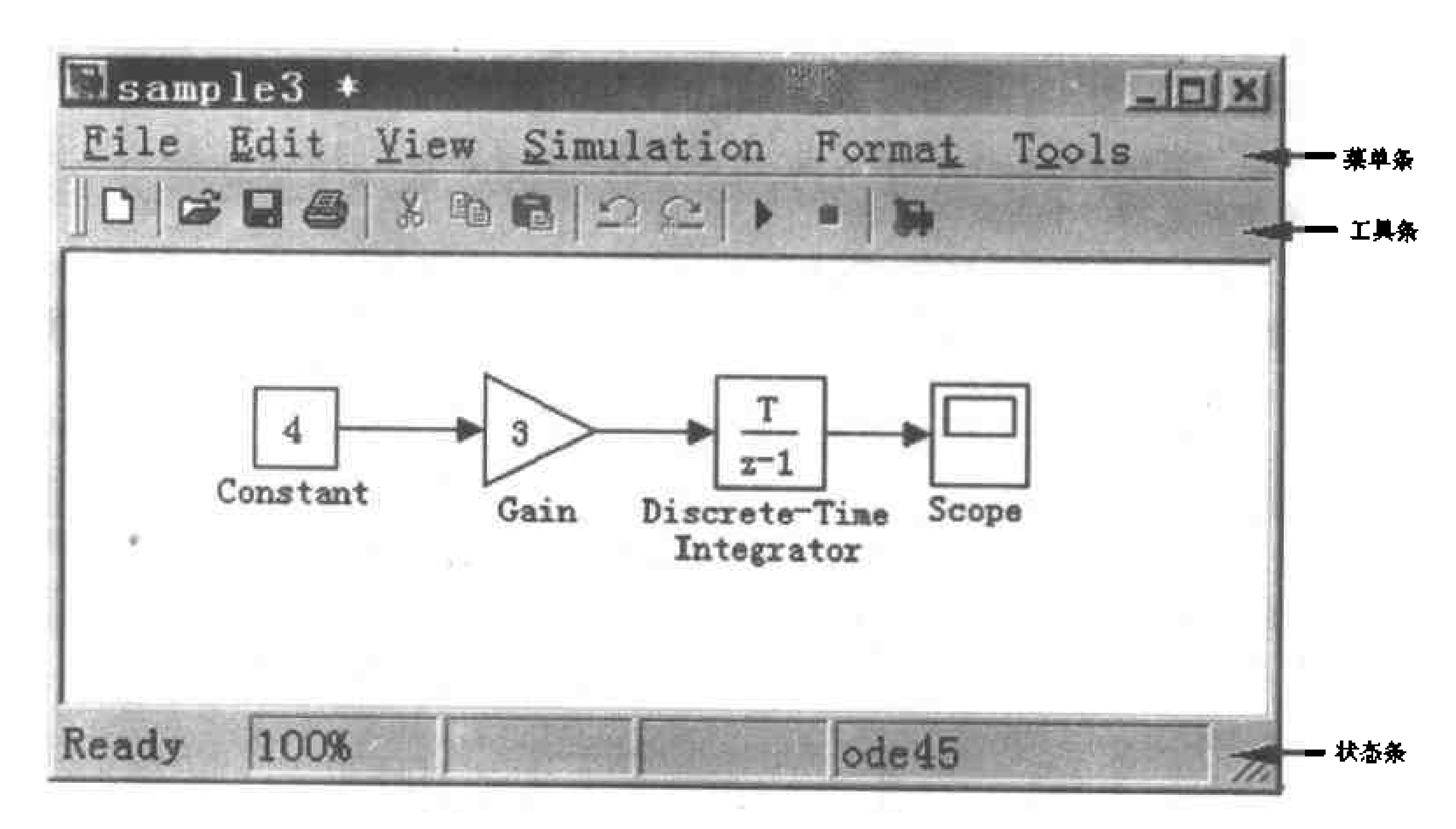


图 2.2 模型编辑窗口

1) 在View 菜单下选择 Zoom In 菜单项,或在模型窗口时按 R 键,就可放大模块视图.

2) 在View 菜单下选择 Zoom Out 菜单项,或在模型窗口时按 V 键,就可缩小模块视图.

3) 在View 菜单下选择Fit System to View 菜单项,或在模型窗口时按空格键,就可使框图正好符合窗口的大小.

4) 在View 菜单下选择Normal 菜单项,就可观察到框图的实际大小.

缺省状态下,无论在模型浏览方框内,还是在一个分开的窗口中,打开一个框图,Simulink 调整模块框图适合于观察的窗口.如果改变一个框图的缩放设置,在关闭框图时,Simulink 就会保存设置,以便下次打开框图时恢复该设置.在下次打开框图时,如果想恢复缺省设置,选择View 菜单下的Fit System to View 菜单项即可.

2.2 选择对象

建模操作中,诸如拷贝一个模块或者删除一条连线,都需要首先选择一个或多个模块或连线,这些模块或连线就叫做对象.

(1) 选择对象

单击对象,就可以选择它.小黑四方块的"句柄"会显示在被选中模块的四个角上,或在被选中的连线的两个端点旁.图 2.3 显示了一个被选中的 Sine Wave 模块和一条被选中的连线:

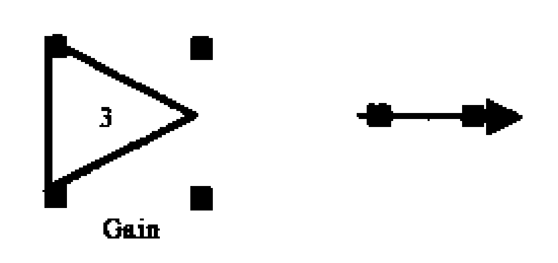


图 2.3 选择对象后显示

用在对象上单击的方法来选择它时,所有其它以前被

选中的对象都会恢复为未选中状态.

(2) 选择多个对象

可以通过一次一个地选择,或者使用一个范围框选择互相邻近的多个对象,或者通过选择整个模型而选择多个对象.

1) 一次一个地选择多个对象. 要通过单独地选择每一个对象来选择多个对象,按下 Shift 键的同时单击每一待选择的对象. 要取消一个已被选择的对象的选中状态,按下 Shift 键的同时再一次单击该对象.

2) 使用范围框选择多个对象. 在窗口的同一区域选择多个对象的一个简单方法则是在对象的周围画一个范围框. 图 2.4 示出用范围框选择对象,其结果如图 2.5 所示.

① 通过定位鼠标指针在一合适的位置来定义范围框的起始角,然后按下鼠标左键并保持住. 注意这时的鼠标指针变成十字形状.

② 拖动指针到范围框的对角位置. 一个点线框将包含被选择的模块和连线.

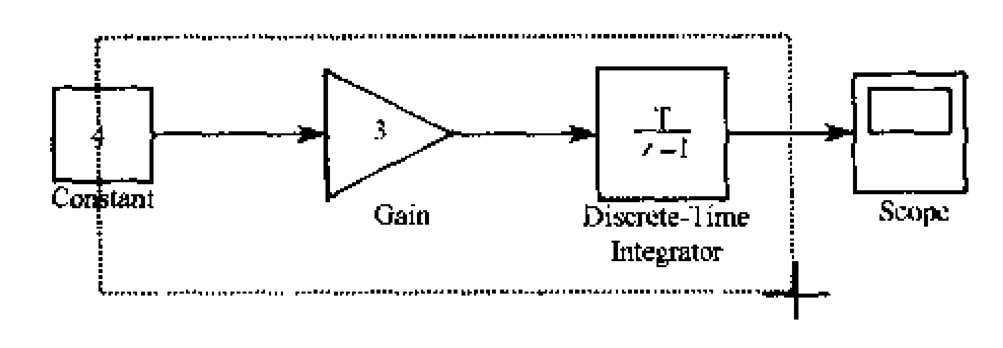


图 2.4　用范围框选择对象

③ 松开鼠标左键. 至少有一部分在范围框内的模块和连线都被选中.

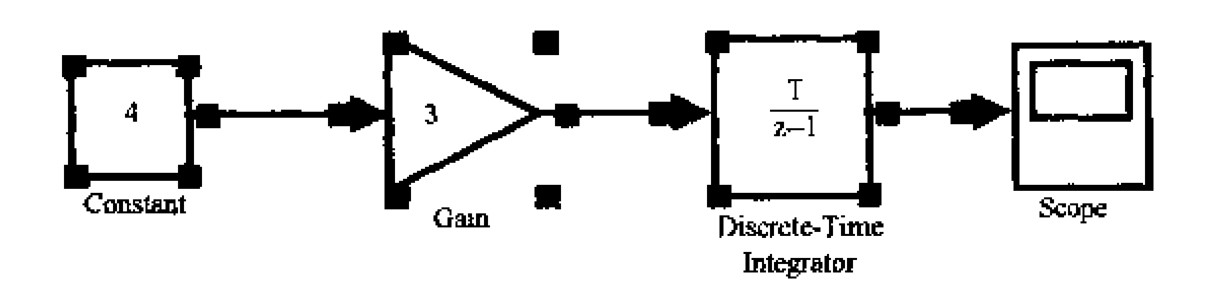


图 2.5　图 2.4 用范围框选择对象后的结果

3) 选择整个模型. 要选择一个活动窗口的所有对象,选择Edit 菜单下的Select All 菜单项即可. 不能通过此种方法,选择所有模块和线来创建子系统模块.

2.3　模　　块

模块是 Simulink 模型构造的基本元素,采用适当的方法创建模块,并使其互相连接,可以建造虚拟的或动态的系统模型. 这一节将介绍如何使用模块建造动态系统模型.

2.3.1　模块数据提示

在 Windows 平台上,当在框图中移动鼠标至一个模块时,Simulink 在一个弹出式窗口显示关于该模块的信息,图 2.6 为移动鼠标至"离散时间积分"模块时的弹出式提示窗口. 要关闭这一特性或者控制数据提示所包含的信息,可改变 Simulink 的View 菜单中的Block Data Tips 下的选择项.

2.3.2 虚拟模块

创建模型时必须了解,Simulink 模块分为两个基本类型:非虚拟模块和虚拟模块.非虚拟模块在系统仿真中起主动作用;如果在模型中添加或删除一个非虚拟模块,模型的运行状态就会改变.相反,虚拟模块在仿真中不起主动作用;它只是以图形方式简单地帮助组织一个模型.一些 Simulink 模块在某些环境下是虚拟的,而在另外一些环境中是非虚拟的,这些模块被称为条件虚拟模块.表 2.1 列出了 Simulink 的虚拟模块和条件虚拟模块.

DiscreteIntegrator:"Discrete-Time Integrator"
Input Port 1:
Output Port 1:
IntegratorMethod=ForwardEuler
ExternalReset=none
InitialConditionSource=internal
InitialCondition=0
LimitOutput=off
UpperSaturationLimit=inf
LowerSaturationLimit=-inf
ShowSaturationPort=off
ShowStatePort=off
SampleTime=1

图 2.6 模块(Discrete-Time Integrator)提示弹出式窗口

表 2.1 Simulink 虚拟模块和条件虚拟模块

模块名	虚拟模块条件
Bus Selector	永远是虚拟的
Data Store Memory	永远是虚拟的
Demux	永远是虚拟的
Enable Port	永远是虚拟的
From	永远是虚拟的
Goto	永远是虚拟的
Goto Tag Visibility	永远是虚拟的
Ground	永远是虚拟的
Inport	除了该模块驻留于一个条件执行的子系统中,并且直接与模块输出相连,其它情况下均是虚拟的
Mux	永远是虚拟的
Outport	如果该模块驻留于任何子系统模块(条件的或非条件)中,并且不是驻留于 Simulink 顶层窗口,那么就是虚拟的
Selector	永远是虚拟的
Subsystem	如果该模块不是条件执行的,那么就是虚拟的
Terminator	永远是虚拟的
Test Point	永远是虚拟的
Trigger Port	如果没有输出端口,那么就是虚拟的

2.3.3 从一个窗口拷贝和移动模块到另一个窗口

建立模型时,会经常从 Simulink 的模块库、其它库或者模型窗口中拷贝模块到建模窗口.要拷贝模块,按如下步骤进行:

1）打开合适的模块库或模型窗口；

2）拖动要拷贝的模块到目标模型窗口．要拖动一个模块，将光标定位于模块的图标上，按下鼠标左键并保持住，移动光标到目标窗口，然后松开鼠标左键．

也可以从 Simulink 库浏览窗口中拖放模块至模型窗口．

要注意的是，当从 Simulink 模块库中拷贝 Sum，Mux，Demux 和 Bus Selector 模块到模型中时，Simulink 会隐藏它们的名称．这样做是为了避免模型图中不必要的混乱，并且这些模块的形状清楚地表明了它们所代表的功能．

还可以通过Edit 菜单下的Copy 和Paste 命令拷贝模块：

1）选择要拷贝的模块；

2）从Edit 菜单下选择Copy 菜单项；

3）击活目标模型窗口；

4）从Edit 菜单下选择Paste 菜单项．

Simulink 自动为每一个拷贝的模块命名．如果它是模型中的第一个该类模块，则它的名字与源窗口中的名字相同．例如，如果从数学库中拷贝 Gain 模块到模型窗口中，则这一新模块的名字将是 Gain．如果模型中已经包含一个名为 Gain 的模块，Simulink 将在以后的模块名字末加进次序号（如 Gain1，Gain2）．用户也可以修改模块的名字．

当拷贝一个模块时，新模块继承了原始模块的所有参数值．

Simulink 使用一个不可见的间隔五象素的栅格以简化模块之间的排列．一个模型内的所有模块都会压住栅格中的一条线．在选择模块后，可以通过按方向键，对模块进行上、下、左、右的微移．

要想在模型窗口显示网格，可以在 MATLAB 窗口输入以下命令：

```
set_param('<model name>','showgrid','on')
```

要想在模型窗口隐藏网格，可以在 MATLAB 窗口输入以下命令：

```
set_param('<model name>','showgrid','off')
```

要改变网格间隔，输入以下命令：

```
set_param('<model name>','gridspacing',<number of pixels>)
```

如要改变网格间隔为 40 象素，输入以下命令：

```
set_param('<model name>','gridspacing',40)
```

上述任何一个命令，可以在选择模型后，输入 gcs 代替'<model name>'．

可以使用 Copy、Cut 和 Paste 等命令拷贝或移动模块到其它兼容的应用程序（诸如 Word 字处理程序）．这些命令只拷贝代表模块的图形，而不拷贝它们的参数．

从一个窗口拖动模块到另一个窗口其实是从一个窗口拷贝模块到另一个窗口，如果拖动时按住了 Shift 键，将会使模块从一个窗口移到另一个窗口．

可以使用Edit 菜单下的 Undo 命令取消加进的模块．

2.3.4 在模型中移动模块

在一个模型窗口中，将单个模块从一个地方移到另外一个地方，只要拖动模块到新的位置就行了．Simulink 将会自动地重画与被移动的模块相连的连线．

要移动多个模块及其连线：

1）选中要移动的模块和连线；

2）将被选中的目标拖到新的位置然后松开鼠标按钮.

2.3.5 在模型内复制模块

可以通过下述方法在模型内复制模块：

按住 Ctrl 键的同时，用鼠标左键选中模块后，将其拖动到一个新的位置.也可以使用鼠标右键做到这一点.复制的模块具有与原始模块相同的参数值.新的模块名被加进了次序号.

2.3.6 指定模块参数值

Simulink 用户界面允许设置模块的参数值，有些模块参数是所有模块都具有的.模块功能的某些方面由模块的参数确定，可以使用模块属性(Block Properties)对话框来设置这些参数；要显示对话框，首先选择要设置属性的模块，然后从 Simulink 的Edit 菜单中选择 Block Properties... 菜单项.图 2.7 所示为模块属性设置对话框.

Block Properties: Discrete-Time Integrator

Info

'Description' is a text field associated with the block that is generally used for saving comments about about the block usage within the model.
'Priority' can be empty or an integer value which specifies the block's sequencing during execution relative to other blocks with priorities in the same window.
'Tag' is a general text field which is saved with the block
'Open function' is the function to be called when you double-click on a block
'Attributes format string' sets the display format shown below the block name (e.g. Pri=%<Priority>).

Properties

Description:

Priority:
3

Tag:

Open function:

Attributes format string:
Priority=%<priority>\nSample Time=%<sampletime>

OK | Cancel | Help | Apply

图 2.7 模块属性设置对话框

其它特定模块参数是特殊的，可使用一个模块特殊参数对话框来设置这些参数，双击一个模块可打开该对话框；图 2.8 所示为模块参数设置对话框。打开该对话框后，可以接受显示的参数值，也可以改变它们的值。

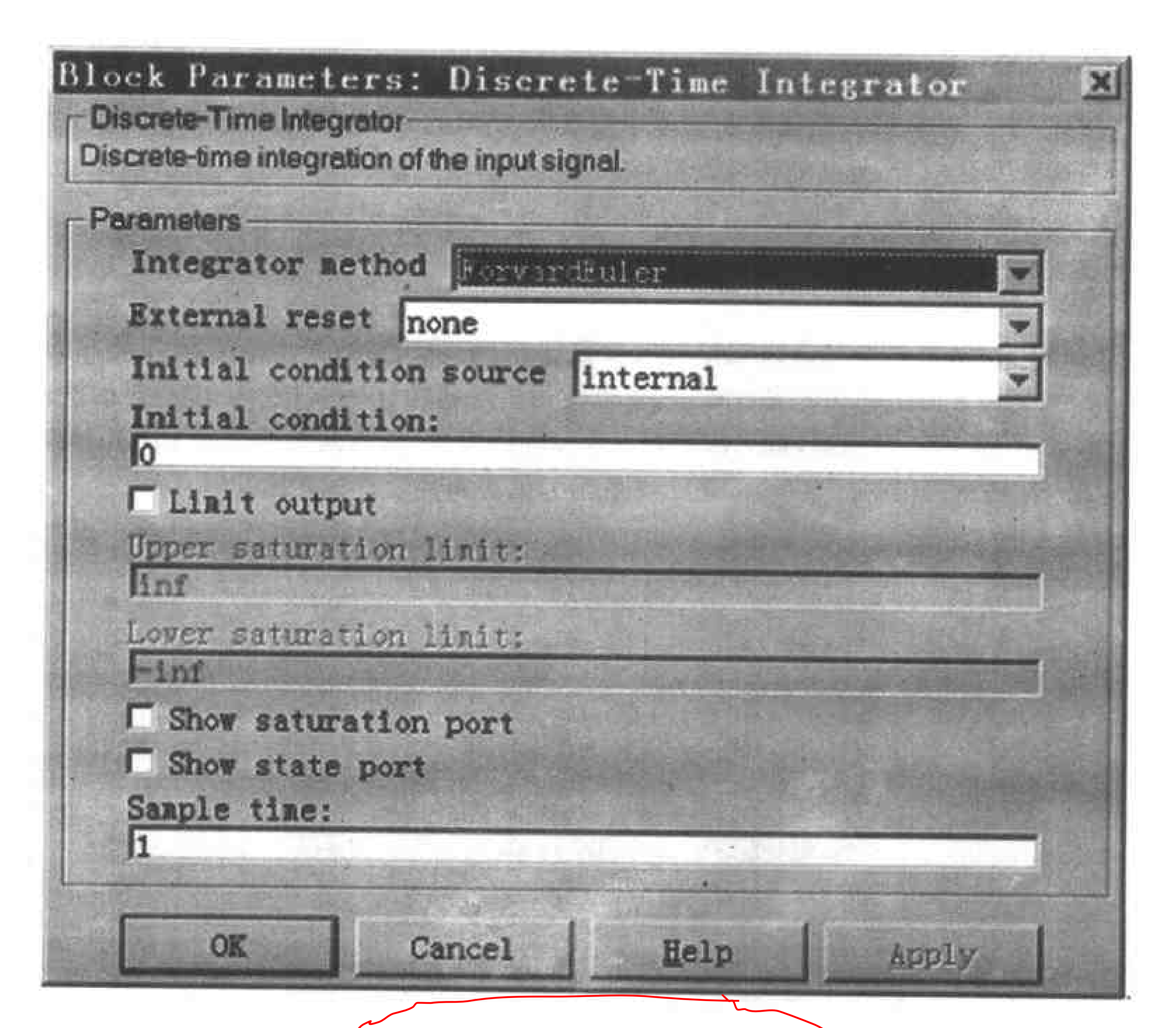


图 2.8　模块参数设置对话框

另外，也可以使用 set_param 命令改变模块的参数值。有些模块对话框，允许指定一些或所有参数的数据类型。

2.3.7　模块属性对话框

使用模块属性对话框可以设置模块的公用参数，如图 2.7 所示。该对话框包含如下域：

1) Description（描述），简要描述模块的作用。

2) Priority（优先次序），在模型中相对其它模块的执行优先次序。

3) Tag（标记），一个随模块一起保存的普通文本字段。

4) Open function（打开函数），当用户打开该模块要调用的 MATLAB(m-)函数。

5) Attributes format string（属性格式串），模块 AttributesFormatString 参数的当前值。该参数指定哪些参数显示在模块图标下。属性格式串可以是任意的含有参数名称的文

本字符串. 其中的参数名称以两“%<”符号开头，以“>”符号结束. 如，其中含有%<priority>. Simulink 在模块图标下显示属性格式串，用相应的参数值替换每个参数名. 可以用换行符“\n”在不同的行显示不同的参数. 如在图 2.7 所示的模块属性对话框的属性格式串中输入：

Priority=%<priority>\nSample Time=%<sampletime>

则该模块将显示为如图 2.9 所示的样式.

如果参数值不是一个字符串或整数，Simulink 在该参数值上显示 N/S(不支持)；如果参数名不合法，Simulink 将显示“???”.

$\frac{T}{z-1}$

Discrete-Time
Integrator
Priority=3
Sample Time=1

图 2.9　模块属性对话框设置后模块显示情况

2.3.8　删除模块

要删除一个或多个模块，选中将被删除的模块，然后按 Delete 或 Backspace 键进行删除. 也可以从Edit 菜单下选择 Clear 或者 Cut 命令. Cut 命令会将模块写到剪贴板上，因此，可以将它们粘贴到一个模型中，而使用 Delete 或 Backspace 键或 Clear 命令则不能做到这一点.

可以使用Edit 菜单下的 Undo 命令恢复被删除掉的模块.

2.3.9　改变模块方向

缺省情况下，信号从模块的左端传到右端. 输入端口在左边，输出端口在右边. 可以通过选择 Format 菜单下的如下菜单项来改变模块的方向：

1）选择 Flip Block 菜单项，将模块旋转 180°；

2）选择Rotate Block 菜单项，将模块顺时针方向旋转 90°.

图 2.10 显示了使用 Flip Block 和Rotate Block 菜单项后，Simulink 是如何排列模块的端口的.

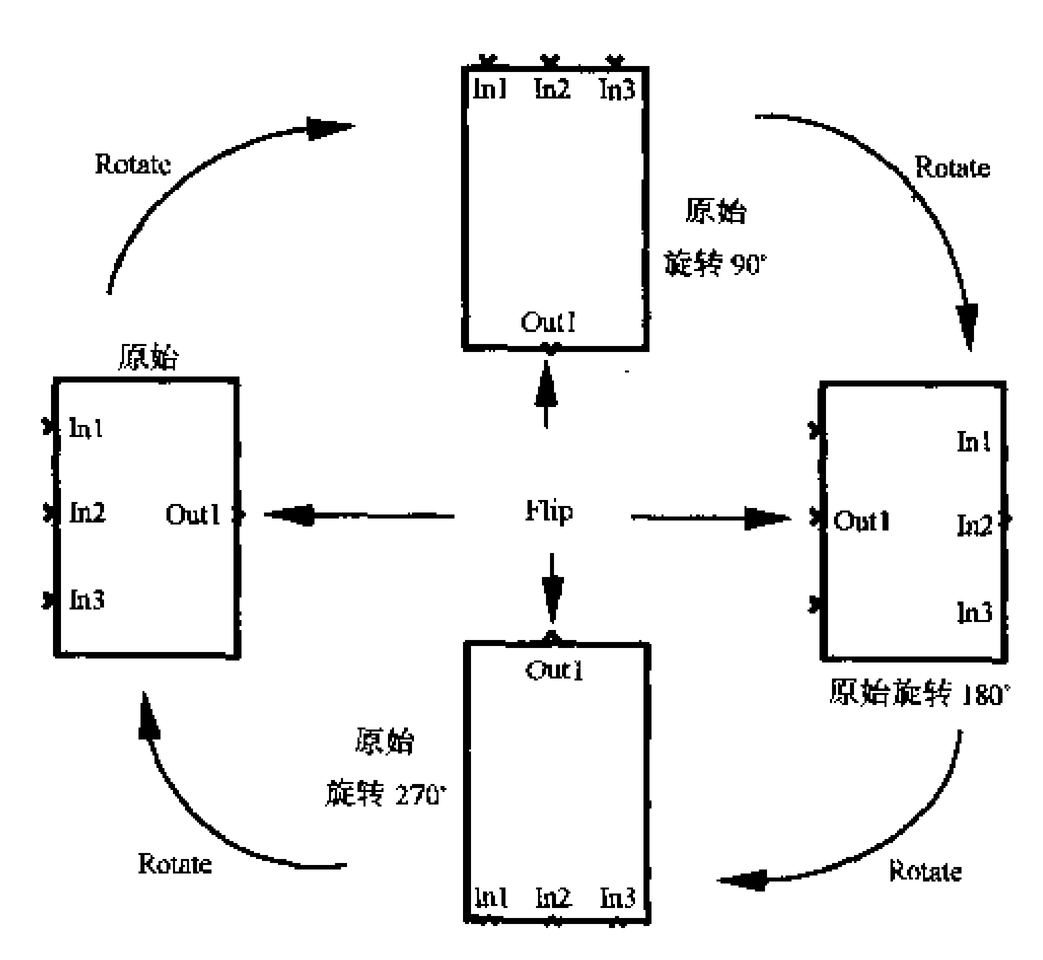


图 2.10　Flip Block 和 Rotate Block 菜单项使用效果

2.3.10　调整模块大小

要调整模块的大小，首先选中它，然后拖动它的任何一个选择句柄. 当按下鼠标键时，一个点线矩形框显示了模块的新大小. 当松开鼠标键时，模块的大小得到了调整. 图 2.11 所示是调整离散时间积分器(Discrete-Time Integrator)模块大小过程的示意图，右下角被选择，并拖至光标位置. 当鼠标键松开后，模块就变成调整后的大小，与图中点虚线框大小一致.

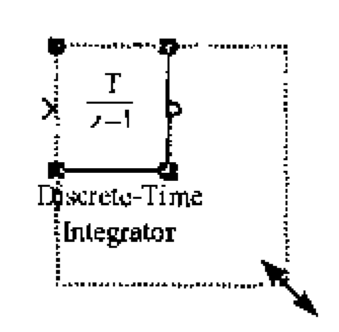


图 2.11　调整模块大小示意图

2.3.11　模块名字处理

模型中所有的模块名都必须是惟一的，并且必须包含至少一个字符. 缺省情况下，如果模块的端口在它的左右两边时，模块的名字显示在它的下面，而如果模块的端口在它的上下两边时，模块的名字显示在它的左边.

2.3.11.1　改变模块名字

编辑模块名字可采用以下方法：

1) 在 Windows 或 UNIX 平台上，要替换模块名字，首先要点击模块名字，然后双击或拖动光标选择整个名字，再输入新名字.

2) 要插入字符，先点击插入点两字符之间，指定插入位置，然后插入文字.

3) 替换字符，拖动鼠标选择替换文字的范围，然后输入新文字.

当在模型的其它地方点击光标，或进行其它任何操作时，就接受或拒绝改变名字. 如果试图将模块名字改变为已经存在的名字，或者空名字(不含任何字符)，Simulink 会显示出一个错误信息. 也可以通过选择模块，再选择 Format 菜单中的 Font 菜单项，在设置字体(Set Font)对话框中选择字体、字形和大小来修改模块名字体；该过程同时也改变模块图标字体.

通过选择 Edit 菜单中的 Undo Block Name 菜单项，或按 Ctrl＋Z 键，取消对模块名的编辑. 但需要注意的是，如果改变了一个模块库的名字，所有与之相联系的模块都将变成未知的.

2.3.11.2　改变模块名字的位置

改变所选模块名字的位置有两种方法：

1) 将模块名拖至模块相反的一边.

2) 通过选择 Format 菜单中的 Flip Name 命令. 该命令将模块名放置于模块的相反一边.

2.3.11.3　改变是否显示模块名

要改变是否显示所选模块的名字，在 Format 菜单下的：

1) 选择 Hide Name 菜单项，以隐藏一个可见的模块名；当选择 Hide Name 菜单项时，该菜单项将变为 Show Name.

2）选择 Show Name 菜单项，以显示一个被隐藏了的模块名.

2.3.12　显示模块图标下的参数

在模块框图中，可以使 Simulink 在模块图标下显示一个或多个模块参数，按以下方法指定要显示的参数：

1）在模块属性对话框中的 Attributes format string 域内输入一个属性格式字符串.

2）使用 set_param 语句，将模块的"AttributesFormatString"属性设置为一格式串.

2.3.13　向量输入和输出

几乎所有的 Simulink 模块都接受标量或向量输入，产生标量或者向量输出，并且允许用户提供标量或向量参数. 这样的模块将在本书称之为向量化了的.

可以通过选择 Format 菜单下的 Wide Vector Lines 菜单项以确定一个模型的哪些连线传输向量信号. 当这一选项被选取时，传输向量信号的连线将比传输标量信号的连线显得粗一些.

如果要在选择Wide Vector Lines 后改变模型的显示，必须选择Edit 菜单下的 Update Diagram 菜单项，以更新模型的显示. 开始仿真也可以更新模块图的显示.

2.3.14　输入和参数的标量扩展

标量扩展是将一个标量值转换为同样元素的向量. Simulink 对大部分模块的输入或参数都可进行标量扩展.

2.3.14.1　输入的标量扩展

当使用有多个输入端的模块(诸如 Sum 或 Relational Operator 模块)，可以将向量输入和标量输入混合在一起. 此时，标量将扩展成相同元素的向量，而宽度与向量输入相等. 如果多个模块的输入是向量，那么它们包含元素的个数应该相等.

如图 2.12 所示，这一模型具有标量输入和向量输入. Constant1 模块的输入将被标量扩展，以匹配 Constant 模块的向量输入. 它的输入将被扩展为向量[2 2 2 2 2].

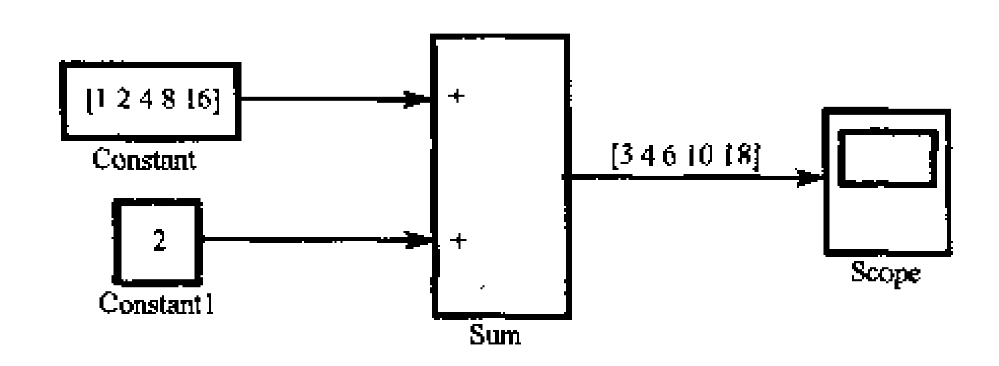


图 2.12　输入标量扩展模型示例

2.3.14.2　参数的标量扩展

可以指定向量化了的模块的参数为向量或者标量. 指定向量参数时，每一个参数元素将与输入向量的相对应的元素相关联. 当指定标量元素时，Simulink 将自动地应用标量

扩展以将它们转换为一个合适长度的向量.

图 2.13 中的例子说明了将一个标量参数(Gain 模块标量参数 2)扩展为一个具有相同元素值的向量([2 2 2 2 2]),以与模块的输入向量相适应.

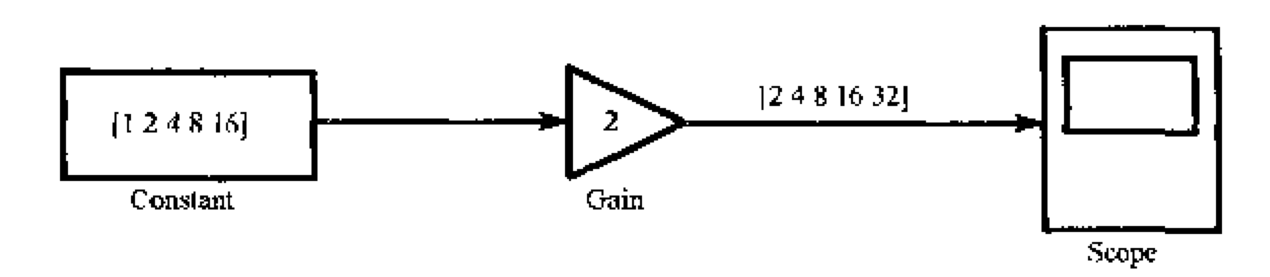


图 2.13 参数标量扩展模型示例

2.3.15 设置模块优先权

可以设置一个模型中的非虚拟模块计算的优先权.高优先权的模块在低优先权的模块之前计算,不必在没有设置优先权的模块之前.可以通过交互或编程方式来设置模块的优先权.编程语句为

```
set_param(b,'Priority','n')
```

其中,b 是模块路径,n 是任意有效整数.负数和 0 都是有效的值.数值越小,优先权越高;即值 3 的优先权大于值 4.交互方式设置模块的优先权时,在模块的属性(Block Properties)对话框中的 Priority 域输入优先权值,如图 2.7 所示.

2.3.16 使用阴影

选取模块,然后选择 Format 菜单下的 Show Drop Shadow 菜单项,可给所选模块加上阴影.当选取了已经被加上了阴影的模块时,该菜单项将变为 Hide Drop Shadow,此时选择该菜单项,将去掉所选模块的阴影.图 2.14 显示了所有模块被加上了阴影的模型.

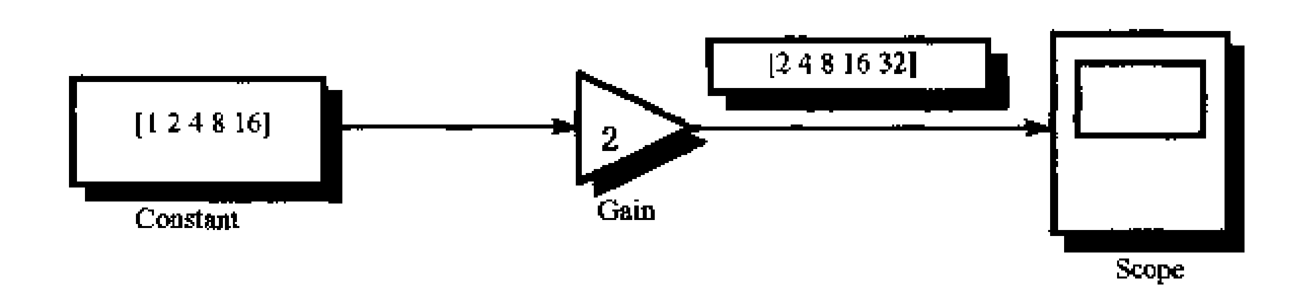


图 2.14 模块加阴影示例

2.4 模块库

模块库使得用户能够从外部模块库中拷贝模块到他们自己的模型中,并且当库中源模块改变时,能够自动地更新被拷贝的模块.无论是开发自己的模块库,还是使用其它人提供的模块库(如模块集),模块库可以确保用户所建的模型,自动包括这些模块的最新版本.

2.4.1 术语

要了解模块库的功能和使用方法，首先必须理解这一功能使用过程中遇到的有关术语.

1) Library(模块库)——库模块的集合. 要创建库，必须用File 菜单下的New 命令下的 Library 菜单项.

2) Library block(库模块)——库中的一个模块.

3) Reference block(引用模块)——库模块的一个拷贝.

4) Link(连接)——引用模块和库模块之间的联系，它可以使得库模块改变后，Simulink 能够更新引用模块.

5) Copy(拷贝)——从一个库模块或者另外的一个引用模块中，创建一个引用模块的操作.

图 2.15 显示了这些术语之间的关系.

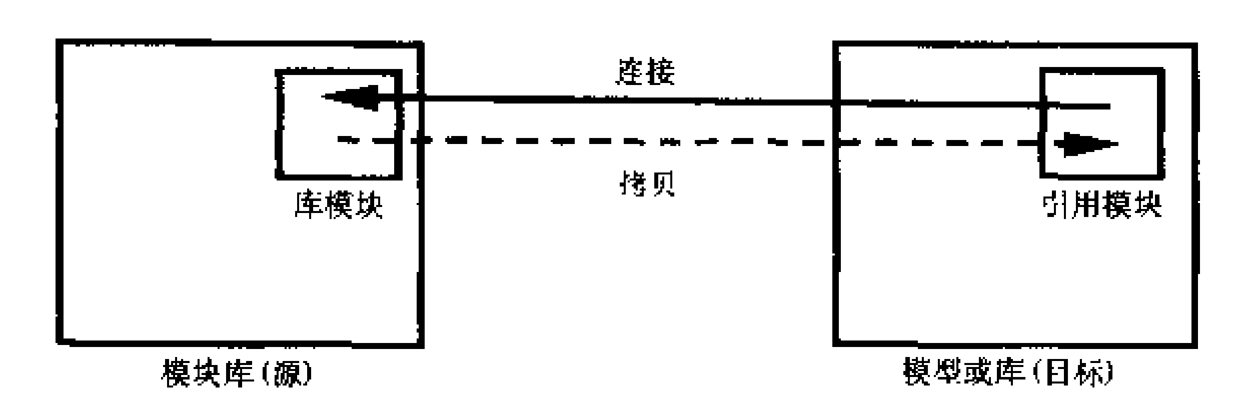


图 2.15 模块库有关术语之间的关系

2.4.2 库的创建与修改

要创建库，选择File 菜单中New 子菜单下的 Library 菜单项后，Simulink 会显示一个名为"Library: untitled"新的窗口. 如果已经存在一个未命名(untitled)的窗口，将会在untitled 末尾加上序号 1，再创建时，untitled 末尾加上序号 2，以此类推.

创建库的另一种方法，从 MATLAB 的命令窗口中输入如下的命令：

```
new_system('newlib', 'Library')
```

这一命令创建一个名为"newlib"的新库. 要显示该库，用命令 open_system.

库必须命名和保存以后，才能从库中拷贝模块.

库打开时，会被自动地锁住，这时修改不了它的内容. 库要解锁，要从Edit 菜单下选择Unlock Library 菜单项. 关闭库的窗口时将会锁住库.

2.4.3 拷贝库模块到模型

可以从库窗口中拷贝、粘贴或拖放模块到模型窗口，也可以从库浏览窗口拖放模块到模型窗口. 当拷贝一个库模块到一个模型或库中时，Simulink 将创建一个到库模块的连接. 引用模块是库模块的一个拷贝，其参数值是可以改变的，但不能对它加模板(mask)，如果它已经加了模板，不能对模板进行编辑. 同样，不能对引用模块设置回调(callback)参

数. 如果在模板方式下浏览引用模块,则 Simulink 显示库模块的顶层系统.

库和引用模块通过名字进行连接,也就是说,引用模块连接到具体的模块和库,它们的名字在拷贝时将会起作用.

如果 Simulink 试图更新引用模块时,在 MATLAB 的路径下找不到库模块或者源库,其连接将会成为未定的(unresolved). Simulink 将会发出一条错误信息,并且用红色的虚线显示模块. 错误信息是:

Failed to find block "source-block-name"
in library "source-library-name"
referenced by block
"reference block-path"

以下任意一种方法可以定位一个未定连接.

1) 删除没有连接的引用模块并且重新拷贝库模块到模型中.

2) 在 MATLAB 的路径(path)中加进所需要的库所在的目录(directory),并且选择 Edit 菜单下的 Update Diagram 菜单项.

3) 双击引用模块. 在出现的对话框中,更正路径名,并点击 Apply 或者 Close 按钮.

所有的模块都有一个 LinkStatus 参数用来表明这一模块是否是引用模块. 这一参数的取值如表 2.2 所示.

表 2.2 LinkStatus 参数取值

参数值	含义
'none'	表明模块不是引用模块
'resolved'	表明模块是引用模块,并且连接确定
'unresolved'	表明模块是引用模块,但连接未定

2.4.4 更新连接的模块

Simulink 在下列时刻更新模型或库中旧的引用模块:

1) 当模型或库被装入的时候.

2) 在从Edit 菜单下选择 Update Diagram 菜单项时,或者运行仿真时.

3) 当使用 get_param 命令查询模块的 LinkStatus 参数时.

4) 当使用 find_system 命令时.

2.4.5 断开与库模块的连接

可以断开引用模块与它的库模块之间的连接,使引用模块成为库模块的一个简单拷贝,不再与库模块相连接;改变库模块不再影响该引用模块. 断开对库模块的连接,使用户的模型成为一个不需要库的独立的模型.

要断开引用模块与它的库模块之间的连接,首先选取该模块,然后选择Edit 菜单下的 Break Library Link 菜单项. 还可以在 MATLAB 的命令窗口中,通过如下的命令改变 LinkStatus 参数的值为'none',以断开引用模块与它的库模块之间的连接:

```
set_param('refblock', 'LinkStatus', 'none')
```

还可以用如下的命令保存一个系统，并且断开引用模块和库模块之间的所有连接：

save_system('systemname', 'newname', 'BreakLinks')

2.4.6 查找引用模块的库模块

要查找与引用模块相连接的源库和模块，首先选取该引用模块，然后选择Edit 菜单下的 Go To Library Link 命令. 如果库处于打开状态，Simulink 选取库模块(显示模块的选取手柄)，并且使源库成为活动窗口. 如果库没有打开，Simulink 打开它并且选取对应的库模块.

2.4.7 获取库模块信息

使用 libinfo 命令，获取系统里面引用模块的信息. 命令的格式为

libdata=libinfo(sys)

其中 sys 是系统的名字. 该命令返回一个大小为 $n\times1$ 的结构，其中 n 为 sys 系统中库模块的个数. 该结构中的每一个元素都有四个域：

1) Block，模块的路径；

2) Library，库的名字；

3) ReferenceBlock，引用模块的路径；

4) LinkStatus，连接状态，为'resolved'或者'unresolved'.

2.4.8 浏览模块库

库浏览器使得用户可以快速定位和拷贝库模块至模型中. 库浏览器窗口如图 2.16 所示.

库浏览器只在 Windows 平台上适用. 定位模块，不仅可以通过浏览库浏览器中库的树，也可以通过库浏览器的搜索工具.

1) 浏览模块库树. 库树显示系统中所安装的所有模块库的列表，可以使用鼠标或键盘，伸展和折叠树来显示或隐藏库的内容. 要伸展或折叠树，点击库项目附近，或选择项目，并按住键盘的左右方向键. 使用上下方向键，上下移动树.

2) 搜索模块库. 要找特定的模块，在邻近库查找按钮的编辑域内输入模块名，然后按"查找"(Find)按钮.

3) 打开一个模块库. 要打开一个库，在浏览器右击库项目，Simulink 显示一个"打开库"(Open Library)按钮，选择该按钮，打开该库.

4) 生成并打开模型. 在库浏览的工具条中，选择"新建"(New)按钮来生成模型，选择工具条上的"打开"(Open)按钮，打开一个已存在的模型.

5) 拷贝模块. 要从库浏览器中拷贝模块到模型，在库浏览器中选择模块，拖放所选模块至模型窗口，放于想要生成模块拷贝的地方.

6) 显示关于模块的帮助. 要显示关于一个模块的帮助，在库浏览窗口中右击模块，并选择其后弹出的按钮.

7) 显示关于模块的帮助. 定住库的浏览器，使库浏览器在桌面处于所有窗口之上，在浏览器工具条上，选择图钉(PushPin)按钮.

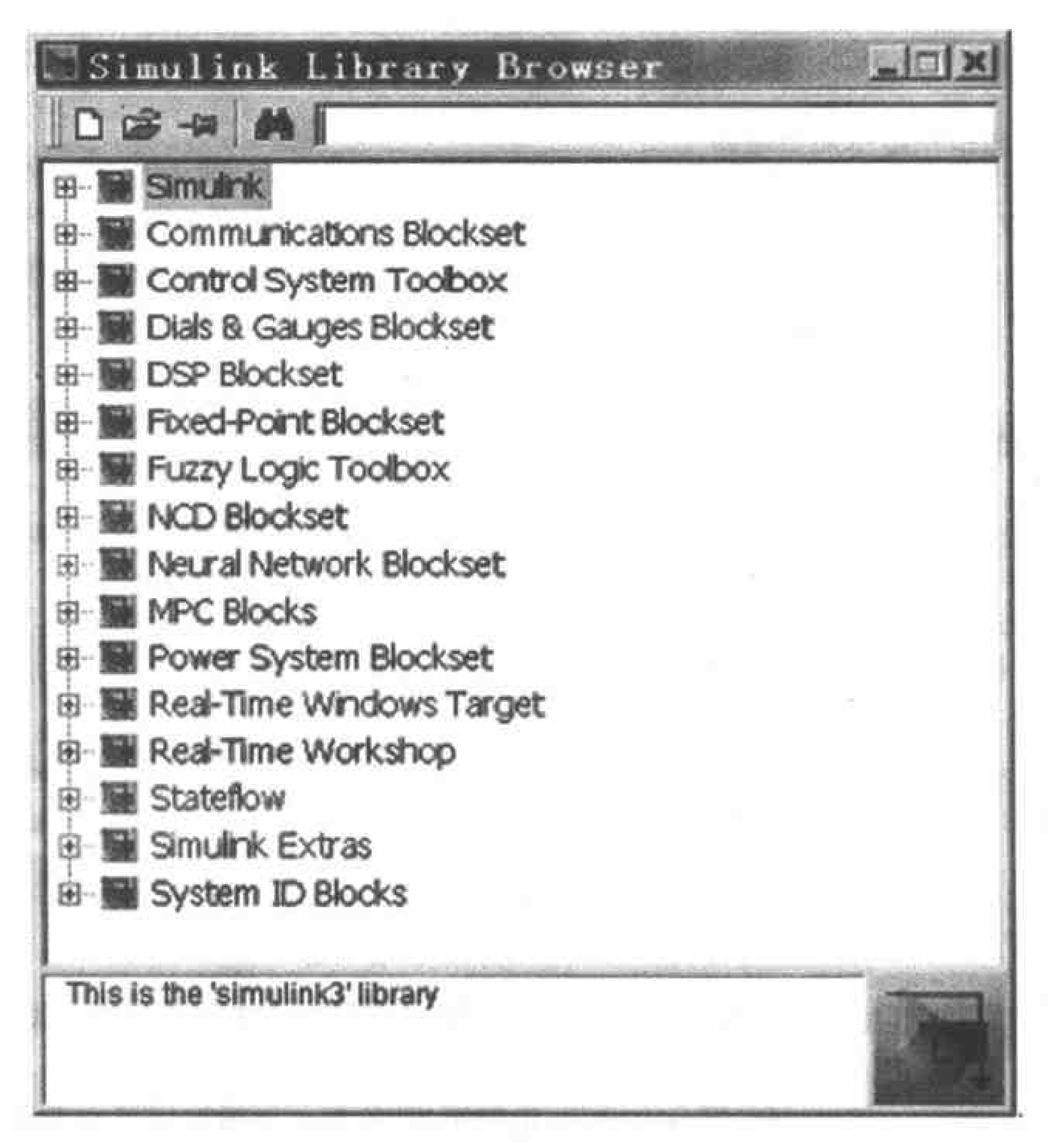


图 2.16　库浏览器窗口

2.5　连接线

信号由连接线传输.连线可传输标量或者向量信号.连线可以将一个模块的输出端口与另一个模块的输入端口相连.连线也可通过支线将一个模块的输出端口与几个模块的输入端口相连.

2.5.1　在模块之间连线

将一个模块的输出端口连到另一个模块的输入端口,具体步骤如下,图 2.17 所示为连接时框图中的显示情况.

1) 鼠标光标置于第一个模块的输出端口,没有必要很精确地将光标放置在端口上.光标的形状将变为细十字形.

2) 按下并保持住鼠标键.

3) 拖动鼠标指针到第二个模块的输入端.可以将光标移到端口上或附近,或者移到

模块内. 如果将光标移到模块内,连线将连接最近的输入端口. 光标的形状变为双十字形.

4) 松开鼠标键. Simulink 将用一条带箭头,以表明信号传输方向的连线代替原来的端口符号. 可以从输出到输入,或者从输入到输出创建连线. 箭头将画在合适的输入端口,而且信号也是相同的.

Simulink 使用水平的或垂直的线段画连接线. 要画斜线,画线时按住 Shift 键.

另外,要断开模块之间的连线,按住 Shift 键,然后将模块拖到一个新的位置.

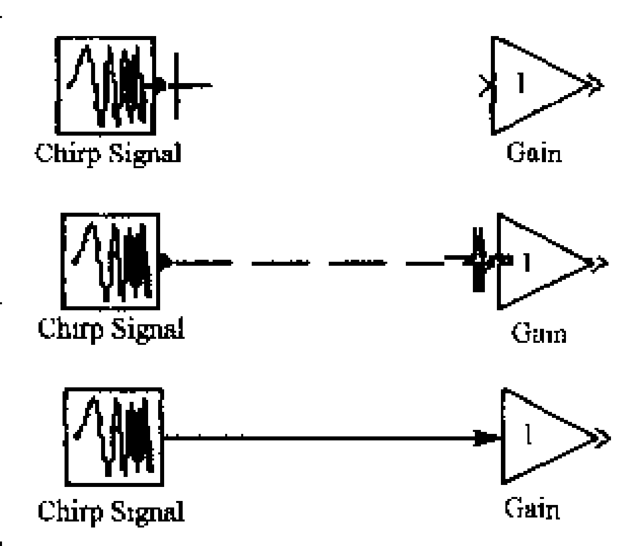


图 2.17 模块之间连线示意图

2.5.2 画支线

支线是从一条已存在的连线开始,将信号传给一个模块的输入端口的连线. 已存在的连线和支线传送的是相同的信号. 使用支线可以将一个信号传给多个模块.

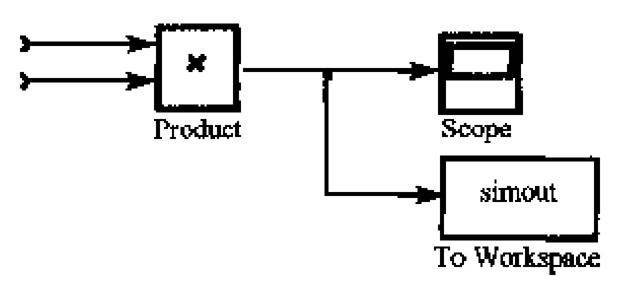


图 2.18 支线示例

在图 2.18 所示的例子中,Product 模块的输出将同时传给 Scope 模块和 Workspace 模块.

要增加一条支线,按如下步骤进行:

1) 将鼠标指针置于要画支线的起点处.

2) 在按住 Ctrl 键的同时(对于 Microsoft Window 和 X Windows)或者 Option 键(对于 Macintosh),按下并保持住鼠标左键.

3) 将鼠标指针拖到目标模块的输入端口,然后松开鼠标键和 Ctrl 或 Option 键.

对于 Microsoft Windows 或者 X Windows,还可以使用鼠标右键来代替 Ctrl 键.

2.5.3 画线段

有时可能希望在某处画一条线段,代替 Simulink 自动画的线. 或者,可能在拷贝一个模块之前,想画一条连到这一模块的连线. 上面两种情况都可以通过画线段来实现.

要画一条线段,可以画一条连线,其末端处于不属于模型图中任何对象所在的区域. 要画另一条线段,将鼠标指针置于该线段的末端,再画另一段. Simulink 画的线段都是水平的或垂直的. 要画斜线段,只要在画线的同时,按住 Shift 键就可以了.

2.5.3.1 *移动线段*

按如下步骤,可移动线段,图 2.19 示出以下步骤.

1) 将鼠标指针置于要移动的线段上.

2) 按住鼠标左键.

3) 拖动光标到希望的位置.

4) 松开鼠标键.

直接与模块端口相连的线段不能够移动.

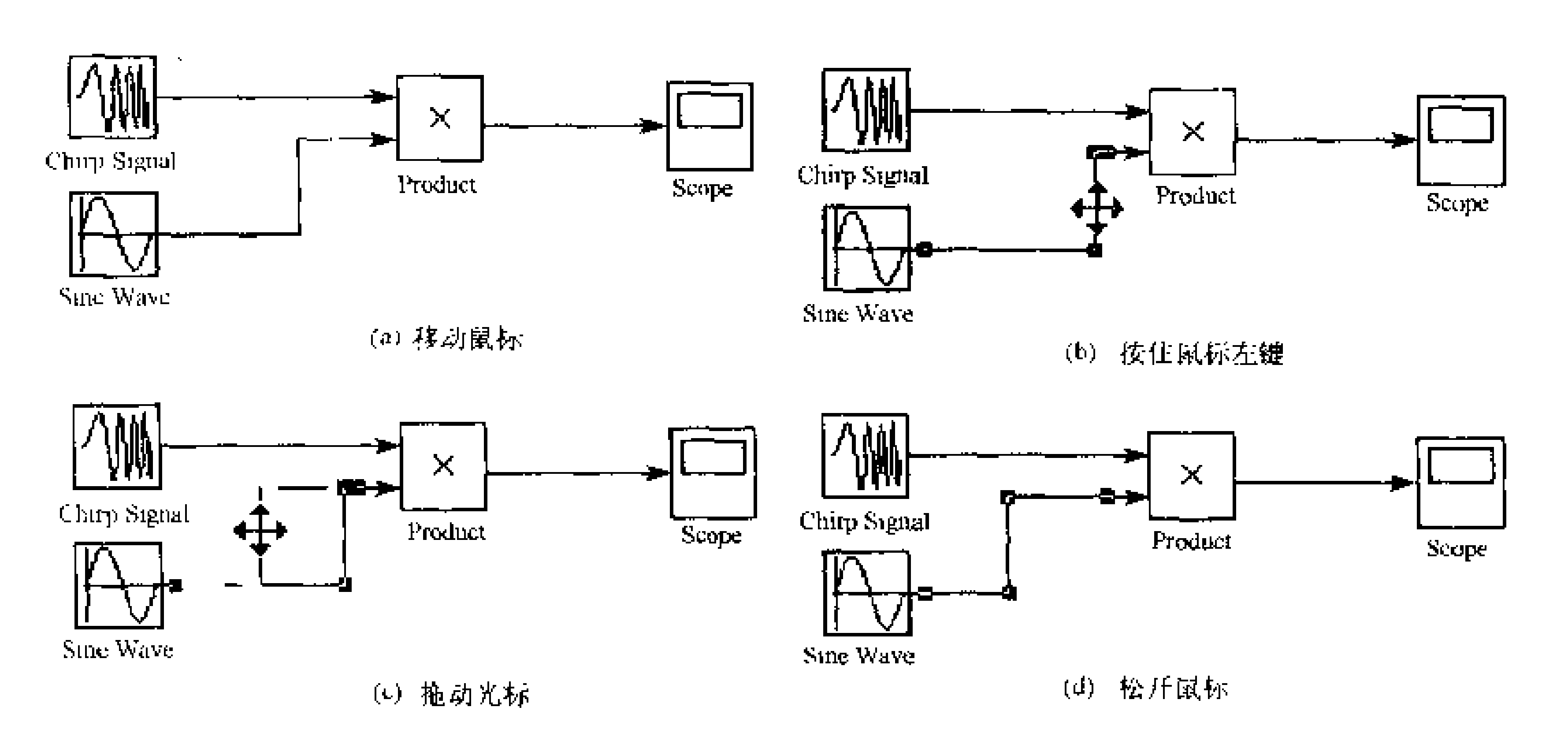


图 2.19 移动线段步骤示意图

2.5.3.2 将一条连线分段

可以将一条连线分成两段，而原来连线的两端还在它们原来的位置. 按如下步骤将一条连线分段，图 2.20 示出以下步骤：

1）选中连线；

2）将指针放于线上想要分断的折点上；

3）按住 Shift 键的同时，按住鼠标左键. 光标的形状将变为包含新的顶点（vertex）的小圆圈；

4）拖动光标到合适的地方；

5）松开鼠标左键和 Shift 键.

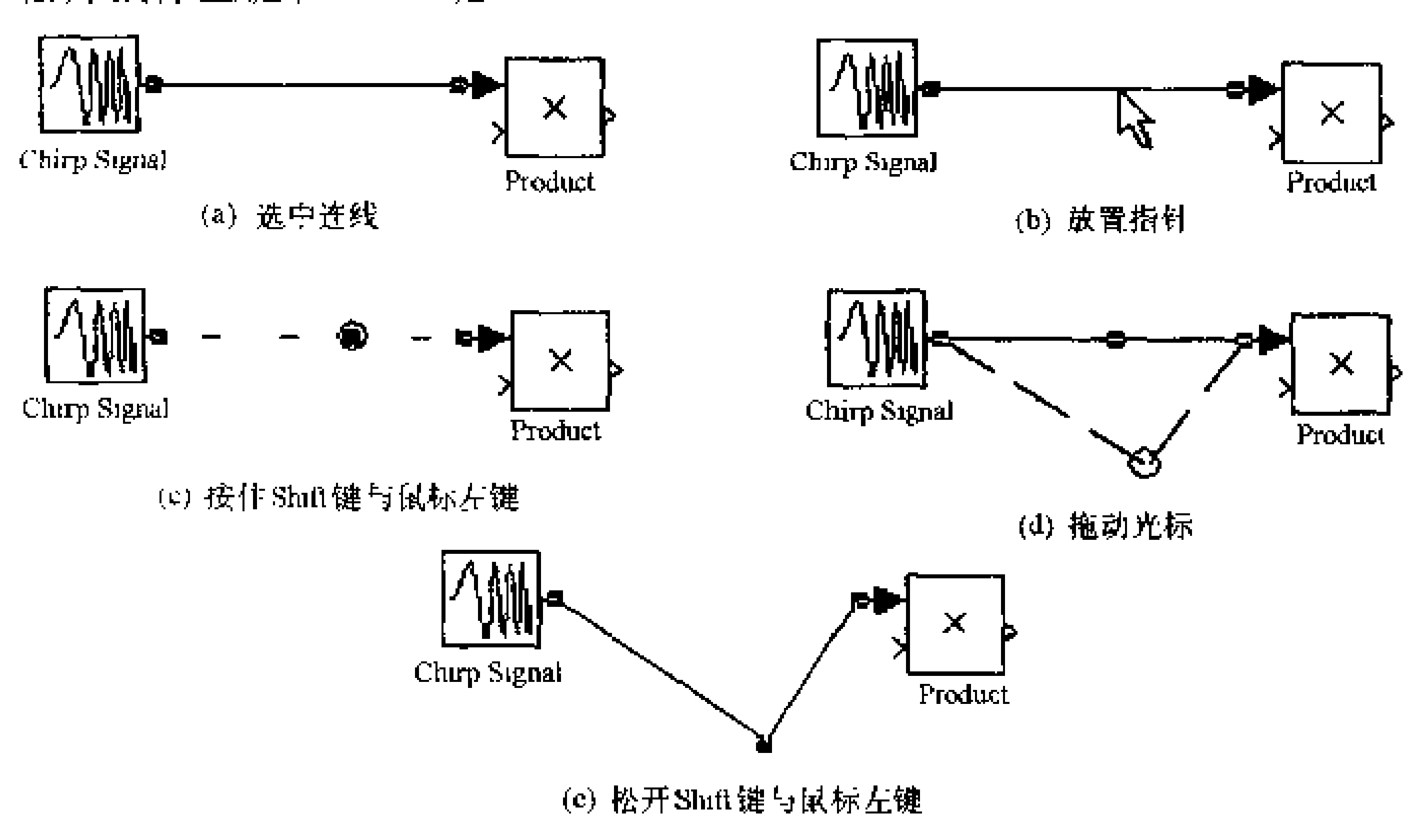


图 2.20 连线分段示意图

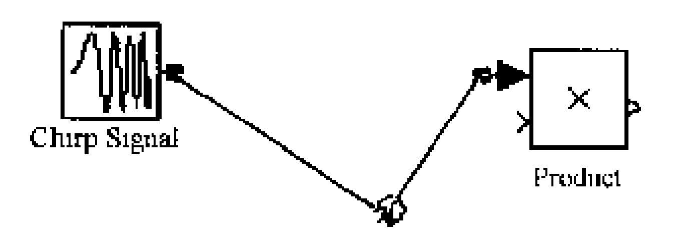
2.5.3.3 移动连线的顶点

要移动连线的顶点，按如下的步骤进行，图 2.21 是其示例.

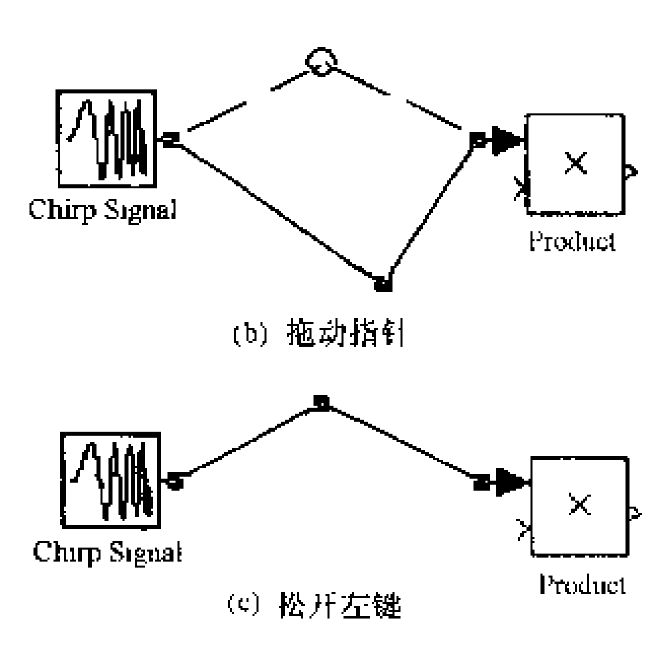
1）将鼠标的指针置于顶点上，然后按住鼠标左键. 这时，光标的形状将变成包含顶点的圆圈；

2）将鼠标指针拖到合适的地方；

3）松开鼠标左键.



(a) 放鼠标指针于顶点上，并按左键



(b) 拖动指针

(c) 松开左键

图 2.21 移动连线顶点示意图

2.5.4 显示连线的宽度

可以通过选中 Format 菜单项下的 Vector Line Widths 菜单项来显示一个模型中所有向量连线的宽度. Simulink 显示模块中每一信号的宽度，包括信号产生模块、信号接收模块. 可以通过选择 Format 菜单项下的 Wide Line Widths 菜单项，用粗线显示向量线.

每启动一次仿真或者更新一次模型图，Simulink 都会探测输入和输出端口的不匹配错误，并且显示出错误信息和模型中连线的宽度.

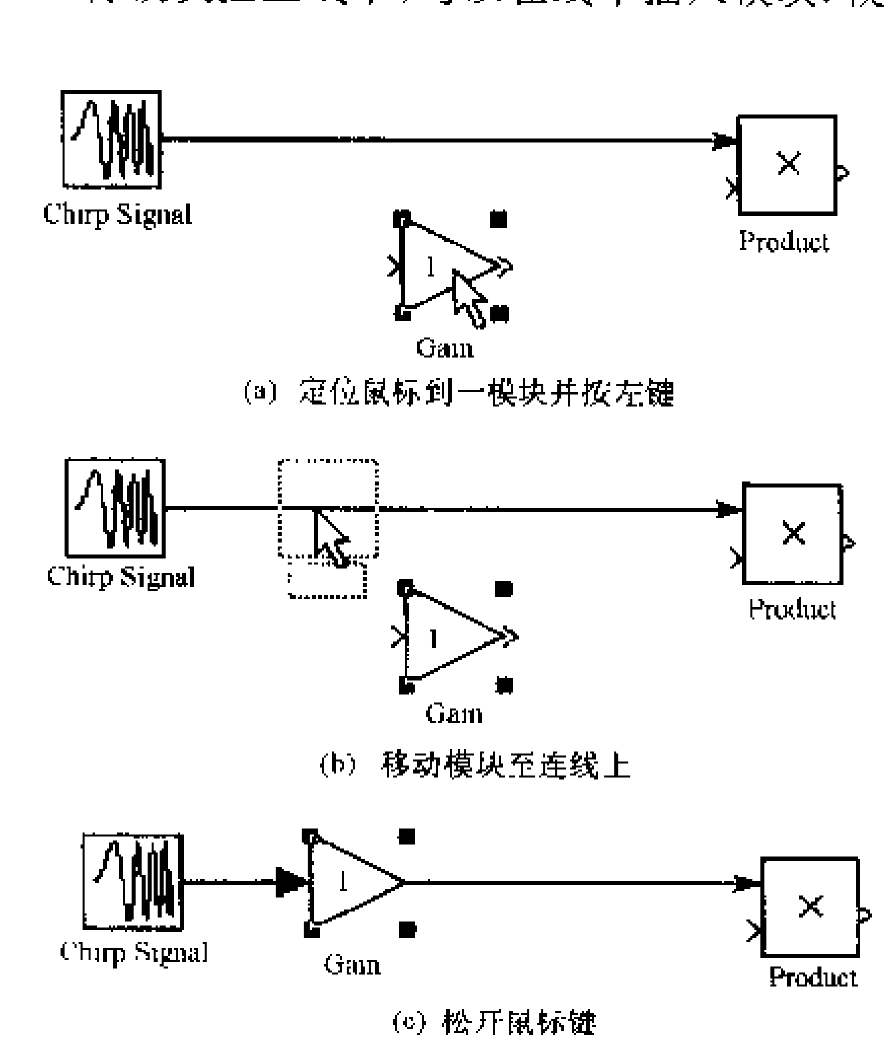
2.5.5 在连线中插入模块

将模块拉至线中，可以在线中插入模块. 模块拉到哪里，就插入到哪里. 而要插入的模块只能有一个输入和一个输出. 插入模块于连线中的步骤如下，如图 2.22 所示.

1）将鼠标定位至一个模块，并按鼠标左键；

2）将模块拉至想要插入的连线上；

3）松开鼠标键，在连线上放下模块，则模块将插入线上.



(a) 定位鼠标到一模块并按左键

(b) 移动模块至连线上

(c) 松开鼠标键

图 2.22 插入模块示意图

2.5.6 信号标注

通过对信号进行标注(label)来对模型注释. 对于水平连线或线段，标注可以出现在其上方或者下方；而对于垂直的连线或线段，标注可以出现在其左边或者右边. 标注可以出现在任一端点、中点处，或者是这些位置的组合.

2.5.6.1　使用信号标注

要建立一个信号标注，只需双击线段，并且在插入点处键入标注即可. 当点击模型的其它部分，标注将会自动定位. 值得注意的是，在建立信号标注时，一定要确保是在连线上双击鼠标，如果在靠近连线但不属于连线的区域双击鼠标，创建的将会是模型的注释.

要移动信号标注，拖动标注到连线上一个新的位置. 松开鼠标键后，标注将会定位于靠近连线的某个地方.

要拷贝信号标注，在拖动标注到连线上的另外一个位置时，按住 Ctrl 键(在 Macintosh 系统中用 Option 键). 松开鼠标键后，标注将同时显示在原来的位置和新的位置上，只能在同一条线上拷贝.

要编辑信号标注，先选取它，然后进行以下几种操作：

1) 要替换标注，先点击标注，再双击或拖动光标选取整个标注，然后输入新的标注.

2) 要插入字符，在两个字符中间点击鼠标以定位插入位置，然后插入文本.

3) 要替换其中的一些字符，拖动光标以选取要替换的文本，然后输入新的文本.

要删除某个信号标注及其所有拷贝，首先删除该标注的所有字符，然后在标注的外部点击鼠标，这时，该标注及其所有拷贝都将会被删除掉. 而要删除标注在某一处的显示，在选取该处标注时，按住 Shift 键，然后按 Delete 或 Backspace 键即可.

要改变信号标注的字体，选取信号后，从 Format 菜单下选择 Font 菜单项，然后在 Set Font 对话框中选择一种字体、字形和大小.

2.5.6.2　信号标注的传播

信号标注的传播也就是自动标注一条从连接模块引出的连线. 支持信号标注传播的模块有：Demux，Enable，From，Inport，Mux，Selector 和 Subsystem 等模块. 已经被加上了标注的信号必须通过连线进入连接模块，而且被传播的信号必须是在该模块出来的连线上或与之相连的连线上.

要传播一个信号标注，在连接模块的输出端口创建一个以"<"开头的信号标注. 这样在运行仿真或者更新模型图时，实际的信号标注将会显示出来，并且两边将会被尖括号括住. 实际的标注是沿连接模块向回跟踪，直到遇到一个信号标注而得到的.

图 2.23 所示的模型中，在更新模块框图前后都有一个信号标注和一个传播标注，该模块在图 2.23(a)中，进入 Goto 模块的信号被标以"Label"，从与之相连的 From 模块出来的信号被标以"<". 图 2.23(b)显示的是选择 Edit 菜单下的 Update Diagram 菜单项后的同一个模型.

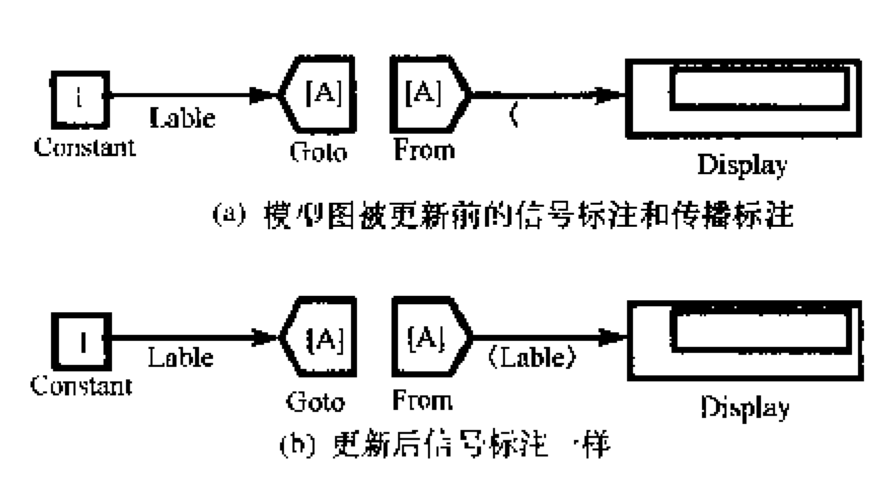


图 2.23　更新模型前后连线标注

在图 2.24 中，传播信号标注显示出了向量信号的内容. 这幅图只显示了模

型图被更新后的样子.

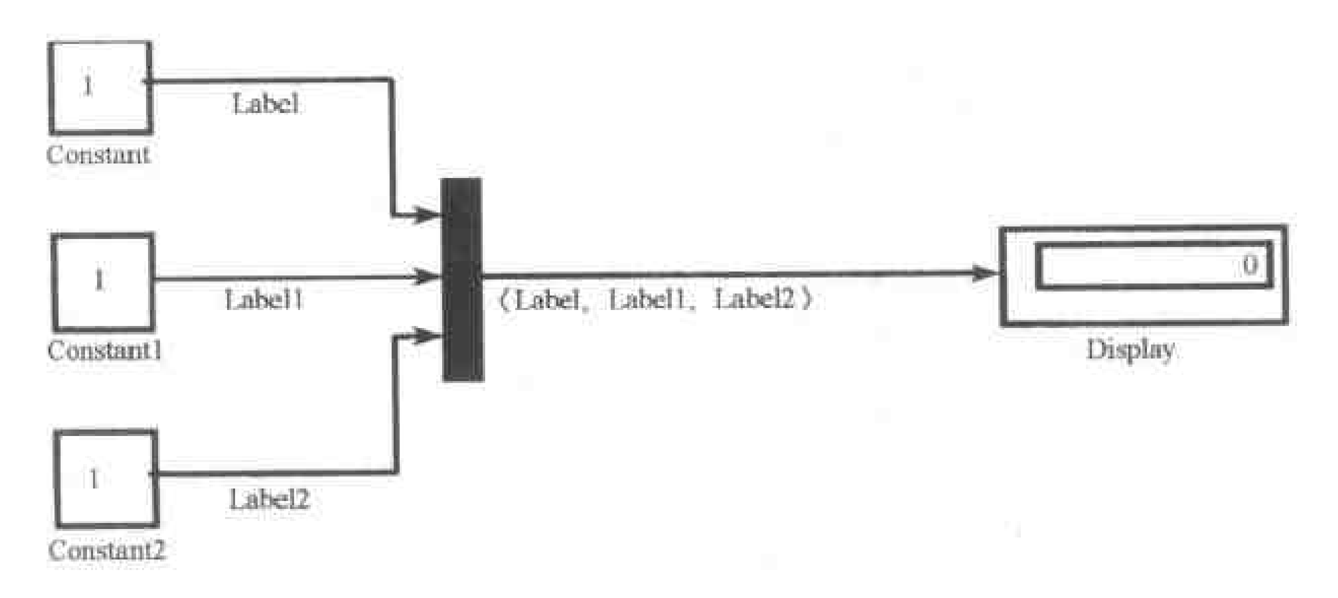


图 2.24　更新模型后，传播向量信号标注示例

2.5.7　设置信号属性

信号都是有属性的，使用 Simulink 的信号属性(Signal Properties)对话框可以浏览或设置一个信号的属性. 选择携信号的连线，从 Simulink 的 Edit 菜单下，选择 Signal Properties 菜单项，即可打开信号属性对话框，如图 2.25 所示.

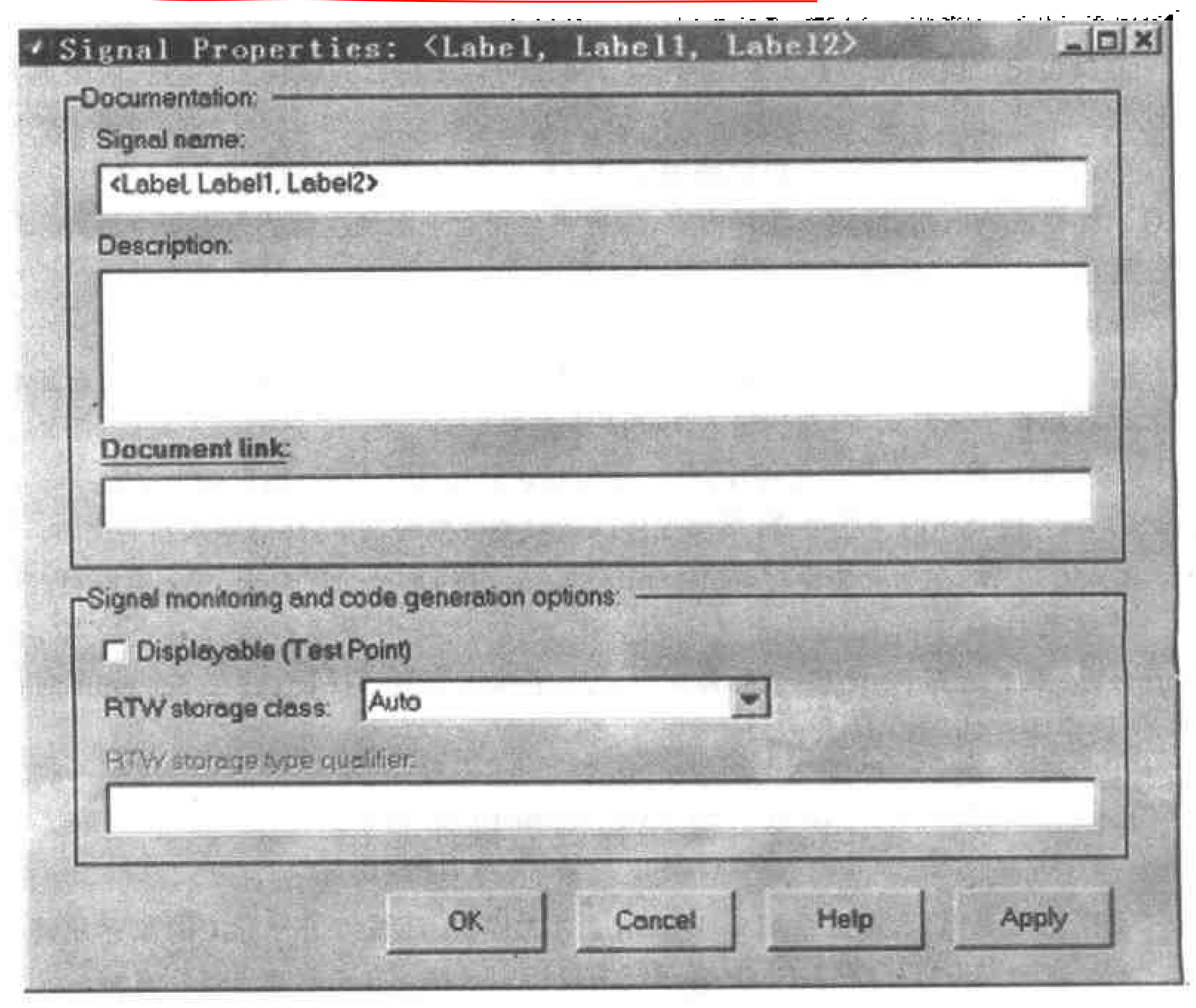


图 2.25　信号属性对话框

2.5.8 信号属性对话框

信号属性对话框允许用户浏览和编辑信号属性. 对话框包括以下控制:

1) 信号名(Signal Name),信号的名称.

2) 描述(Description),在该域中填入信号的描述.

3) 文件连接(Document Link),在该域中输入 MATLAB 表达式来显示信号文件. 要显示文件,点击域标注(即"文件连接"). 例如输入表达式:

```
web(['file:///' which ('foo_signal.html')]
```

在该域中,如果点击域标注,使用 MATLAB 缺省 Web 浏览器,显示 foo. signal. htm

4) 可显示(Displayable 测试点). 检查该选项表明,在仿真期间,信号可以显示出来. 值得注意的是,下两个控制用来设置 RTW(Real-Time Workshop)从模型产生代码时所要使用的属性. 如果不想从模型产生代码,可以忽略它们.

5) RTW 存储类(RTW storage class),从列表中选择信号存储类.

6) RTW 存储类型限定(RTW storage type qualifier),从列表中选择信号存储类.

2.7 注　　释

注释为一个模型提供了文字性的信息. 用户可以在自己的模块图中的任何空地方加上注释.

要在模型中新建注释,在模块图中空白区域双击鼠标,这时将会出现一个小的矩形,并且光标变成插入指针. 输入注释内容,每行将会相对于围绕注释的方框居中对齐.

要移动注释,将注释拖到一个新的位置即可.

要编辑注释,先选取它,然后可以进行以下操作:

1) 在 Microsoft Windows 或者 UNIX 系统中,要替换注释,先点击注释,再双击或拖动光标选取整个注释文本,然后输入新的注释文本.

2) 要插入字符,在两个字符中间,点击鼠标以定位插入位置,然后插入文本.

3) 要替换其中的一些字符,拖动光标以选取要替换的文本,然后输入新的文本.

要删除注释,在选取注释时,按住 Shift 键,然后按 Delete 或 Backspace 键即可.

要改变注释中所有或部分文字的字体,选取注释中需要改变字体的文本,再从 Format 菜单下选择 Font 菜单项,此时会出现字体设置对话框,然后在 Set Font 对话框中,选择字体、字形和字体大小.

2.8 鼠标和键盘操作总结

下面这些表格(表 2.3 至表 2.6)总结了有关模块、连线和信号标注的鼠标和键盘操作. LMB 表示点击鼠标左键;CMB 表示点击鼠标中键;RMB 表示点击鼠标右键.

表 2.3 列出了应用于模块的一些鼠标和键盘操作.

表 2.3 处理模块的有关鼠标和键盘操作

任 务	Microsoft Windows	Macintosh	UNIX
选取一个模块	LMB	鼠标键	LMB
选取多个模块	Shift+LMB	Shift+鼠标键	Shift+LMB；或只用 CMB
从别的窗口拷贝模块	拖动模块	拖动模块	拖动模块
移动模块	拖动模块	拖动模块	拖动模块
复制模块	Ctrl+LMB 并且拖动；或者 RMB 并且拖动	Option+拖动模块	Ctrl+LMB 并且拖动；或者 RMB 并且拖动
连接模块	LMB	鼠标键	LMB
断开模块连接	Shift+拖动模块	Shift+拖动模块	Shift+拖动模块；或者 CMB 并且拖动模块

表 2.4 列出了应用于连线的鼠标和键盘操作.

表 2.4 处理连线的鼠标和键盘操作

任 务	Microsoft Windows	Macintosh	X Windows
选取一条连线	LMB	鼠标键	LMB
选取多条连线	Shift+LMB	Shift+鼠标键	Shift+LMB；或只用 CMB
画支线	Ctrl+拖动连线；或 RMB 并且拖动连线	Option+拖动连线	Ctrl+拖动连线；或 RMB+拖动连线
使连线绕过模块	Shift+画线段	Shift+画线段	Shift+画线段；CMB 并且画线段
移动线段	拖动线段	拖动线段	拖动线段
移动顶点	拖动顶点	拖动顶点	拖动顶点
创建线段	Shift+拖动连线	Shift+拖动连线	Shift+拖动连线；或者 CMB+拖动连线

表 2.5 列出了应用于信号标注的一些鼠标和键盘操作.

表 2.5 处理信号标注的鼠标和键盘操作

任 务	Microsoft Windows	Macintosh	X Windows
创建信号标注	双击连线后键入文本	双击连线后键入文本	双击连线后键入文本
拷贝信号标注	Ctrl+拖动标注	Option+拖动标注	Ctrl+拖动标注
移动信号标注	拖动标注	拖动标注	拖动标注
编辑信号标注	点击标注文本，然后编辑	点击标注文本，然后编辑	点击标注文本，然后编辑
删除信号标注	Shift+单击标注，然后按 Delete 键	Shift+选取标注，然后按 Delete 键	Shift+选取标注，然后按 Delete 键

表 2.6 列出了应用于模型注释的一些鼠标和键盘操作.

表 2.6 处理模型注释的鼠标和键盘操作

任 务	Microsoft Windows	Macintosh	X Windows
创建注释	双击模块图，然后键入文本	双击模块图，然后键入文本	双击模块图，然后键入文本
拷贝注释	Ctrl+拖动注释	Option+拖动注释	Ctrl+拖动注释
移动注释	拖动注释	拖动注释	拖动注释
编辑注释	点击注释文本，然后编辑	点击注释文本，然后编辑	点击注释文本，然后编辑
删除注释	Shift+选取注释，然后按 Delete 键	Shift+选取注释，然后按 Delete 键	Shift+选取注释，然后按 Delete 键

2.9 创建子系统(Subsystems)

随着模型的大小和复杂性的增加，可以将它的模块分成不同的组，组成若干子系统，以达到简化模型的目的. 使用子系统有如下几个方面的优点：

1) 可以减少模型窗口中显示的模块数量.

2) 可以将功能上有关联的模块放在一起.

3) 可以建立一个层次结构的模块图，子系统模块位于一层，组成子系统的各模块位于另外一层.

创建子系统一般有两种途径：第一，在模型中加入子系统模块，然后打开该子系统模块，并且在子系统窗口中加入它所包含的模块；第二，加入组成子系统的模块，然后将这些模块组成一个子系统.

2.9.1 通过加入子系统模块创建子系统

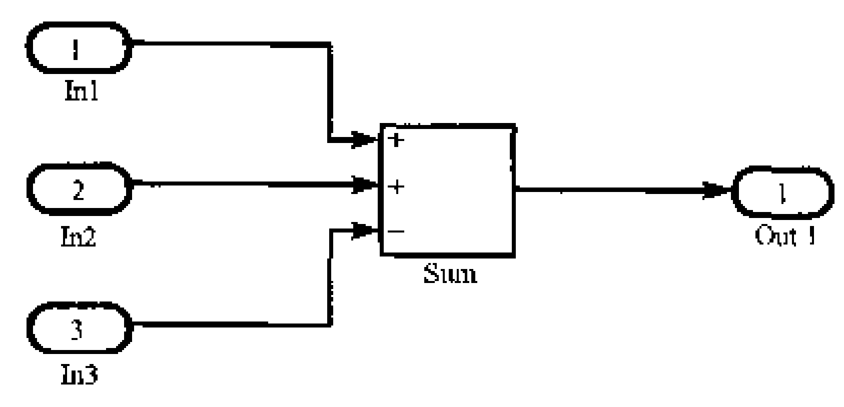
要在子系统模块中加入它所包含的模块之前创建子系统，先在模型中加进一个 Subsystem 模块，然后加入组成该子系统的模块，其过程如下：

1) 从 Simulink 的信号与系统(Signals & Systems)库中拷贝 Subsystem 模块到模型中.

matlab7中为simulink/commonly used blocks下的subsystem

2) 双击该子系统模块，打开它.

3）在空的子系统(Subsystem)窗口中，创建子系统．使用输入(Inport)模块代表该子系统从外部的输入，使用输出(Outport)模块代表该子系统的输出．例如，图 2.26 所示的子系统包含 Sum 模块，并且包含 Inport 模块和 Outport，以表示从该子系统的输入和输出．



双击该模块，可以对SUM模块的输入口个数和形状（圆形或正方形）进行修改

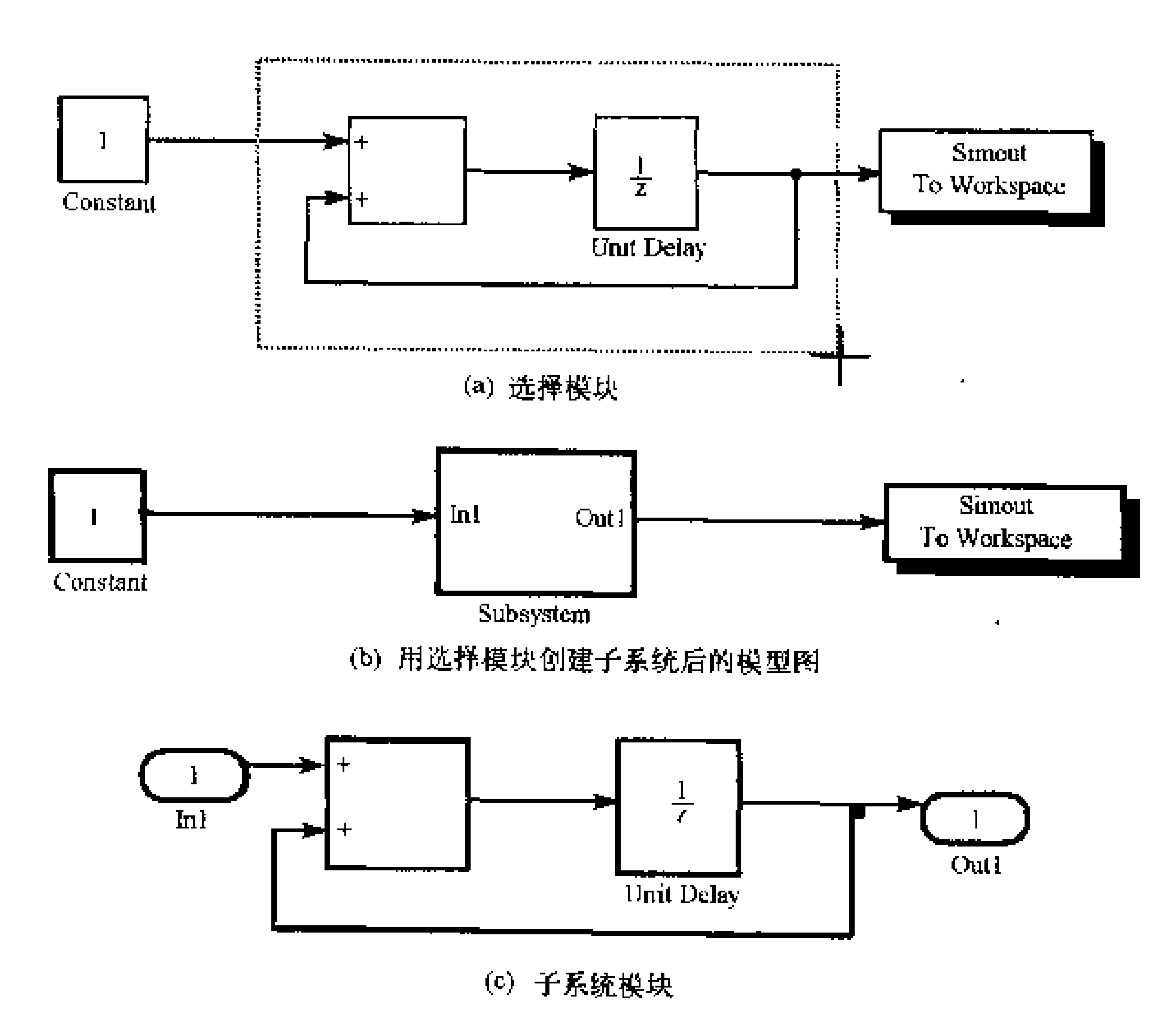
图 2.26　子系统模块及其输入与输出

2.9.2　通过将一些已有模块组织在一起创建子系统

如果模型中，已经包含了组成子系统的模块，可以通过将这些模块组织在一起来创建一个子系统．其过程如下：

1）将子系统需要包含的所有模块和连线用一个边界框包括起来．不能通过单个地选取每一个组成子系统的模块或通过使用 Select All 命令来选取它们．

图 2.27(a)给出了一个代表计数器的模型．Sum 和 Unit Delay 模块被一个边界框包括起来：



(a) 选择模块

(b) 用选择模块创建子系统后的模型图

(c) 子系统模块

图 2.27　创建子系统模块

松开鼠标键时，这两个模块和所有的连接线都被选取了．

2）从 Edit 菜单下选择 Create Subsystem 菜单项，Simulink 将用一个 Subsystem 模块替代被选取的所有模块．图 2.27(b)给出了选择 Create Subsystem 命令后该模型的样子(Subsystem 模块的大小被重新设置了以便能够看清它的端口标注)：

打开 Subsystem 模块，Simulink 显示下一层系统，如图 2.27(c)所示. 注意：Simulink 增加了 Inport 和 Outport 模块以表示该子系统与外界的输入和输出.

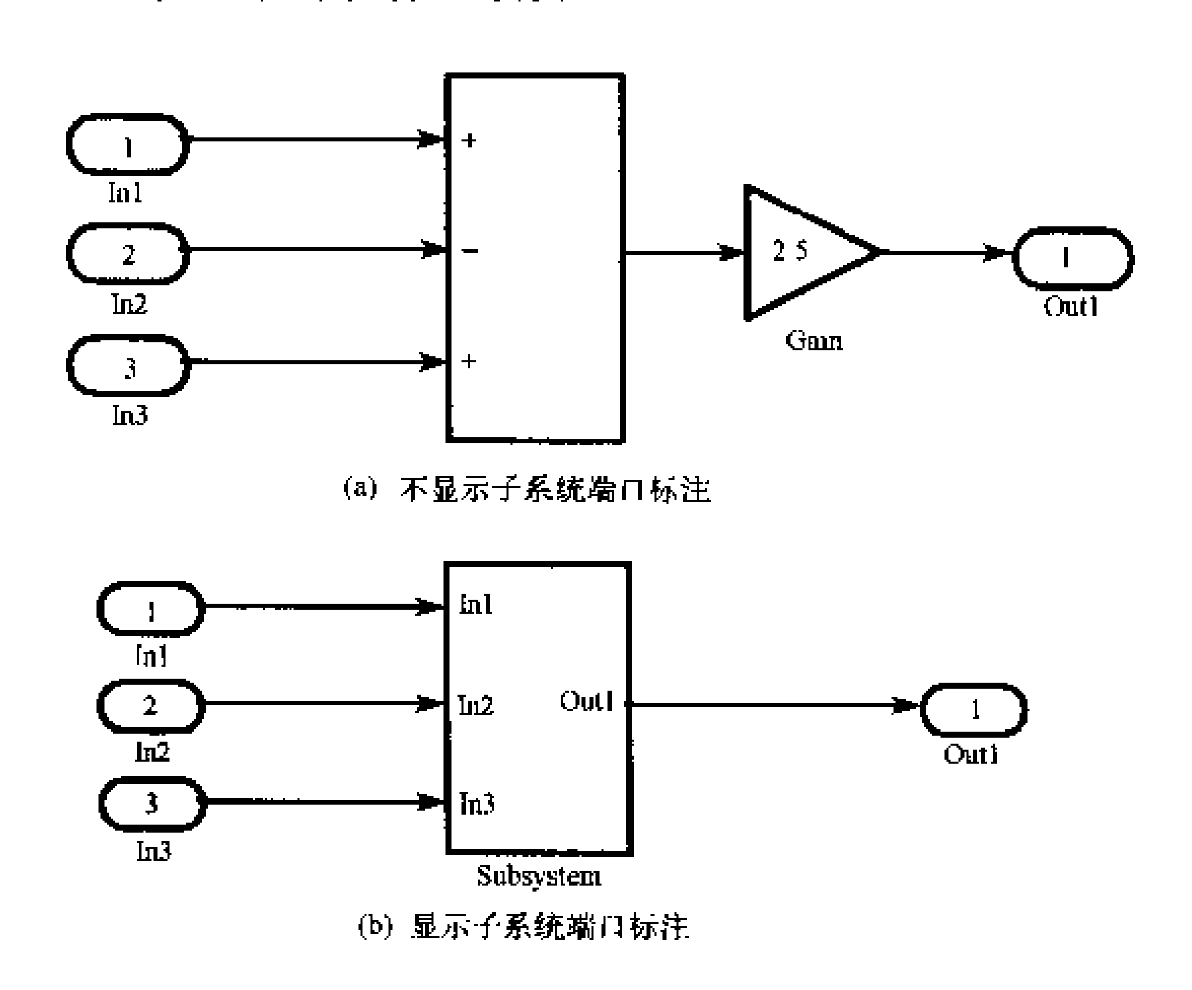
同所有的模块一样，可以改变 Subsystem 模块的名字. 也可以通过模板(masking)特性定制该模块的图标和对话框.

2.9.3 给 Subsystem 模块的端口加上标注

Simulink 会在 Subsystem 模块的端口处加上标注. 这些标注是 Inport 和 Outport 模块的名字，子系统模块是通过这些端口与外面的模块相连的.

可以通过选取 Subsystem 模块，然后选择 Format 菜单下的 Hide Port Labels 命令来隐藏这些端口标注. 也可以通过选取子系统中的一些 Inport 或 Outport 模块后，选择 Format 菜单下的 Hide Port Labels 命令来隐藏一或多个端口标注.

图 2.28 显示了两个模型，左边的子系统包含三个 Inport 模块和一个 Outport 模块. 右边的 Subsystem 模块显示出了端口的标注.



(a) 不显示子系统端口标注

(b) 显示子系统端口标注

图 2.28 具有输入与输出端口的子系统

2.9.4 使用回调程序(Callback Routines)

当对模块图或模块执行特殊的操作时，可以定义需要执行的 MATLAB 表达式. 这些表达式，也称为回调程序，与模块或模型的参数有关. 例如，与模块的 OpenFcn 参数有关的回调程序，将会在模型用户双击模型的名字或改变路径时执行.

要定义回调程序并将它们与参数联系起来，用 set_param 命令.

下面的命令将会在用户双击 DemoModel 模型的 Demo 模块时计算 DemoVar 变量的值：

```
set_param('DemoModel/Demo','OpenFcn',DemoVar)
```

可以查看 Clutch 系统(clutch.mdl)以了解与一些模型回调函数有关的程序. 表 2.7

和表 2.8 列出了可以定义回调程序的参数，并且指明了这些回调程序将在什么时候执行. 在动作之前或之后执行的程序将会在动作之前或之后立即执行.

表 2.7 模型回调参数

参 数	执行时刻
CloseFcn	模块图被关闭之前
PostLoadFcn	模型被装入内存之后. 定义该参数的回调程序对于产生一个要求模型已经被装入内存的界面非常有用
PostSaveFcn	模型被保存之后
PreLoadFcn	模型被装入内存之前. 定义该参数的回调程序对于将模型需要用到的变量装入内存非常有用
PreSaveFcn	模型被保存之前
StartFcn	仿真开始之前
StopFcn	仿真结束之后. 在 StopFcn 被执行之前输出被写进工作空间(workspace)变量或文件之中

表 2.8 模块回调参数

参 数	执行时刻
CloseFcn	模块被用 close_system 命令关闭的时候
CopyFcn	模块被拷贝之后. 该回调程序对于 Subsystem 模块是递归的(也就是说，如果拷贝一个包含已经定义了其 CopyFcn 参数的模块的 Subsystem 模块，该程序也会被执行). 如果用 add_block 命令拷贝模块时回调程序也会被执行
DeleteFcn	模块被删除之前. 这一回调程序对于 Subsystem 模块是递归的
InitFcn	模块图被编译之前，并且在计算模块参数之前
LoadFcn	模块图被装入内存之后. 该回调程序对于 Subsystem 模块是递归的
ModelCloseFcn	模块图被关闭之前. 该回调程序对于 Subsystem 模块是递归的
NameChangeFcn	改变模块的名字或者路径之后. 当一个 Subsystem 模块的路径被改变的时候，它在调用它自己的 NameChangeFcn 程序之后，对于它所包含的所有模块递归地调用该程序
OpenFcn	当模块被打开的时候. 这一参数一般地被用于 Subsystem 模块. 当双击模块或在用这一模块作为一个参数值调用 open_system 命令时，该程序被执行. OpenFcn 参数重载了打开模块时的一般行为，这些一般行为是指显示模块的对话框或打开子系统
ParentCloseFcn	关闭一个包含该模块的子系统之前，或者在用 Create Subsystem 命令将该模块作为一个新的子系统的一部分的时候
PreSaveFcn	模块图被保存之前. 该回调程序对于 Subsystem 模块是递归的
PostSaveFcn	模块图被保存之后. 该回调程序对于 Subsystem 模块是递归的
StartFcn	模块图被编译之后且在仿真开始之前
StopFcn	在仿真被终止的时候
UndoDelete	当删除模块的操作被取消的时候

2.10 创建模型的一些技巧

下面简要介绍创建模型过程中的一些经验，掌握这些经验是相当有用的.

1）内存问题. 一般地讲，内存越大，Simulink 工作的性能越好.

2）采用层次结构. 在比较复杂的模型中，加进层次结构的子系统是很有好处的. 将模块分组可以简化模型的顶层并且使得模型易于阅读和理解.

3）整理模型. 组织合理，并且文字标注得比较清楚的模型易于阅读和理解. 在模型中，必要时加上信号标注和模型注释，有助于描述模型中各部分的作用.

4）建模策略. 如果经常在模型建立过程中，用到相同的一些模块，将这些模块保存在一个模型中，将会简化建模的工作. 在创建新的模型时，只需打开该模型并且从中拷贝要用到的模块. 可以通过选取若干模块的集合组成一个系统，并且保存该系统，以创建一个模块库，以后，就可以在 MATLAB 的命令窗口中输入该系统的名字来使用它.

通常，创建模型时，先在纸上画草图，再用计算机创建. 开始在模型中放进模块时，应在连接它们之间的连线之前，在模型窗口中放进所有的模块，这样，就可以减少打开模块库的次数.

2.11 对方程的建模

对于 Simulink 的新手来说，最令他们费解的问题之一，是如何创建方程的模型. 下面举一些例子来介绍如何创建方程的模型.

2.11.1 将摄氏温度转换为华氏温度的公式模型

将摄氏温度转换为华氏温度的公式是：

$$T_F = \frac{9}{5}T_C + 32 \tag{2.1}$$

其中，T_F 表示华氏温度，T_C 表示摄氏温度.

首先，考虑一下建立模型所需要的模块有：

1）一个 Ramp 模块用来输入温度信号，来自 Sources 库；

2）一个 Gain 模块用来将输入信号乘以 9/5，来自 Math 库；

3）一个 Constant 模块用来定义常量 32，来自 Sources 库；

4）一个 Sum 模块用来将两个数相加，也位于 Math 库；

5）一个 Scope 模块用来显示输出信号，位于 Sinks 库.

第二步，将上面提到的这些模块放进模型窗口中，有必要时，对各模块外观、大小、位置进行调整，如图 2.29 所示.

第三步，分别通过双击 Gain 和 Constant 模块以打开它们的对话框来给它们的参数赋值，输入合适的值后，点击 Close 按钮，应用这些值，并关闭对话框.

第四步，连接各模块，如图 2.30 所示.

Ramp 模块输入摄氏温度，打开该模块并且将其 Initial output 参数的值改为 0；Gain

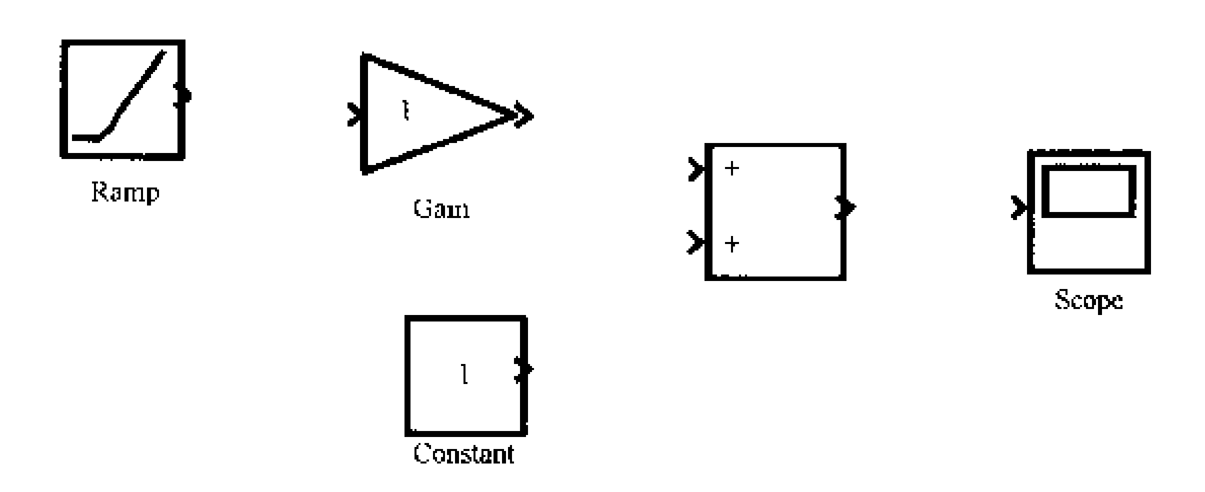


图 2.29 将所需模块放进模型窗口

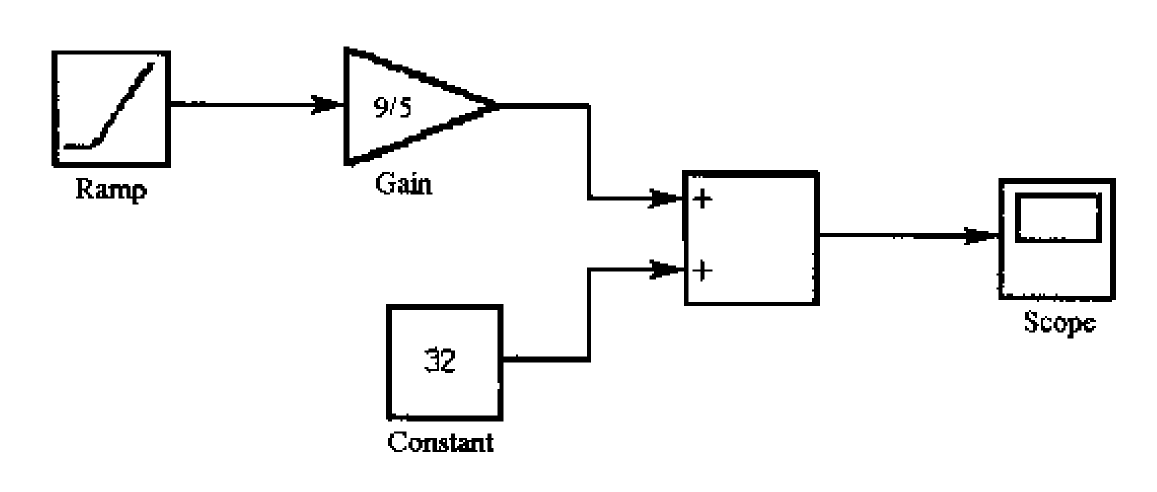


图 2.30 将各模块连接后

模块将该温度值乘以常数 9/5；Sum 模块将结果再加上 32，这样输出结果就是华氏温度了.

打开 Scope 模块以查看输出结果. 现在可以通过选择 Simulation 菜单下的 Start 命令运行仿真了.

2.11.2 创建一个简单的连续系统模型

给式(2.2)的微分方程建模：

$$x'(t) = -2x(t) + u(t) \tag{2.2}$$

式中 $u(t)$ 是一幅度为 1 频率为 1 弧度/分的方波. Integrator 模块将它的输入 x' 求积分以得到 x，在该模型中另外还需要一个 Gain 模块和一个 Sum 模块，要产生方波，使用 Signal Generator 模块并选择 Square Wave 的形式，将其缺省的单位设为弧度/分，这时再一次使用 Scope 模块显示输出. 在模型窗口中放入所有需要的模块并定义好 Gain 模块.

在该模型中，要让 Gain 模块的方向调个头，为此使用 Format 菜单下的 Flip Block 命令，要从 Integrator 模块的输出创建一条支线连到 Gain 模块，画线时按住 Ctrl 键. 现在可以连接所有的模块，如图 2.31 所示.

在该模型中一个重要的概念是包含由 Sum 模块、Integrator 模块和 Gain 模块组成的一个回路. 在该方程中，x 是 Integrator 模块的输出，它也是计算 x' 的模块的输入，这一关系是通过回路实现的.

Scope 显示了 x 的结果，对于一个持续 20 秒的仿真，输出结果如图 2.32 所示.

该例中的方程也可被表示成一个传递函数. 下面的模型中使用传递函数(Transfer Fcn)模块，它接收 u 作为它的输入并且输出 x，所以该模块实现了 x/u. 如果在前面的方

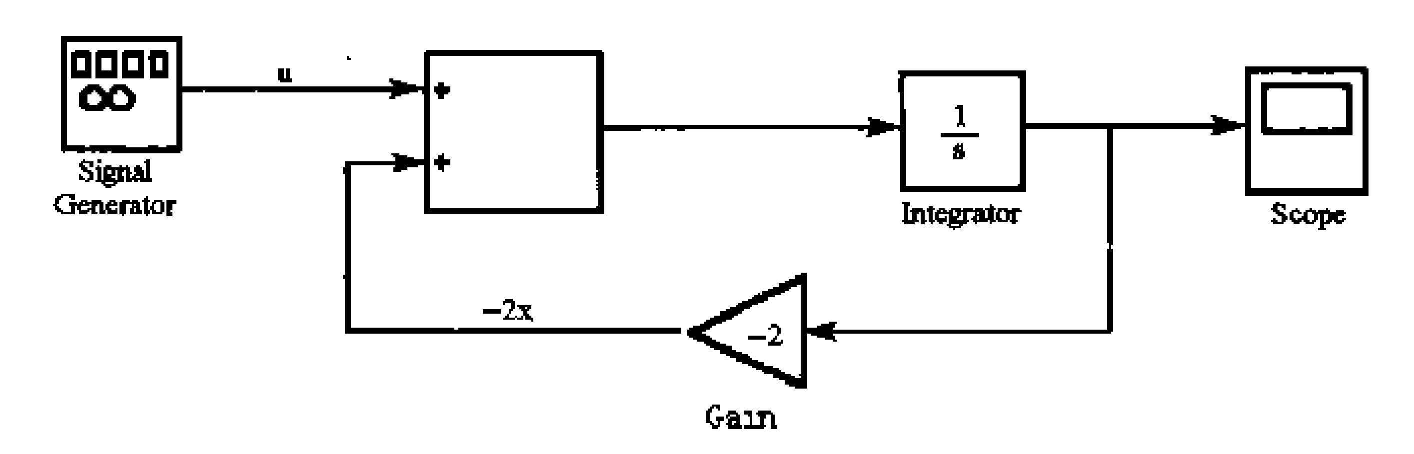


图 2.31 一简单微分方程的模型

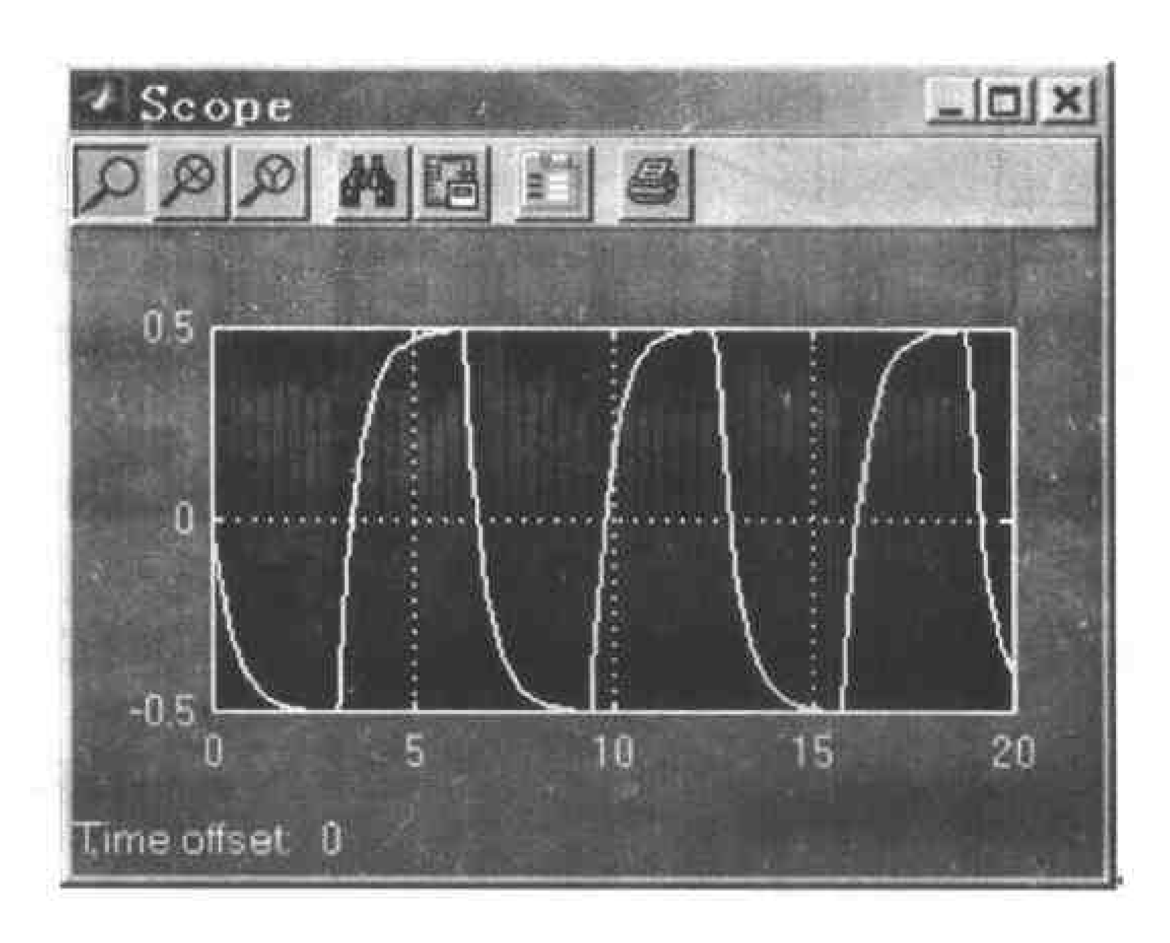


图 2.32 仿真结果显示

程中用 sx 代替 x'，方程将变成

$$sx = -2x + u \tag{2.3}$$

解得 x 为

$$x = u/(s+2) \tag{2.4}$$

或者

$$x/u = 1/(s+2) \tag{2.5}$$

Transfer Fcn 模块中使用参数来指定分子和分母的系数. 在该模型中，分子为 1，分母为 $s+2$. 这两个参数是 s 的连续的按降幂排列的系数向量. 在该模型中，分子为[1]，分母为[1 2]. 这样该模型变得非常简单，如图 2.33 所示. 该模型的仿真结果与前面的完全相同.

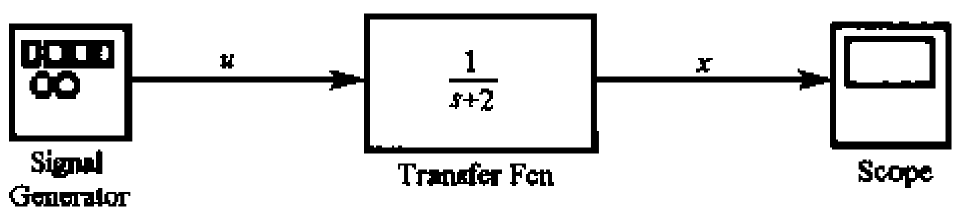


图 2.33 利用传递函数模块创建模型

2.12 数据类型

数据类型也就是计算机在内存中表达数据的方法，一个数据类型决定分配给一个数据的存储空间的多少，决定其二进制编码的方法，还决定可使用的操作与运算. 大多数计算机提供表达数值的数据类型的选择，每一种类型在精度范围、动态范围、性能和使用空间方面都有其特定的优点，利用数据类型的各自优点，可以使得用户能够优化 MATLAB 程序的性能，MATLAB 允许指定 MATLAB 变量的数据类型. 建立在此能力基础上的 Simulink，也允许用户指定 Simulink 信号和模块参数的数据类型. Simulink 的这一能力，在实时控制应用方面相当有用. 例如，在从一个模型生成代码时，通过自动代码生成工具(如 RTW)，允许一个 Simulink 模型指定优化数据类型用于表达信号和模块参数. 通过为模型信号和参数选择最恰当的数据类型，可以使得从模型生成代码的性能大大提高，而空间大大减小. 在仿真前和仿真期间，Simulink 将进行广泛的检查，以保证模型类型安全，即从模型生成代码不会出现溢出和下溢，而产生不正确的结果. 使用 Simulink 缺省数据类型——double(双精度)的 Simulink 模型是类型安全的. 如果不打算从模型生成代码，或者不使用非缺省数据类型的用户，就没有必要考虑数据类型. 相反，如果打算生成代码，并且使用非缺省数据类型，就必须了解数据类型的规则，避免引起数据类型错误，从而阻止模型运行完成和完全仿真.

2.12.1 Simulink 支持的数据类型

Simulink 支持所有 MATLAB 内置数据类型. 表 2.9 给出所有 MATLAB 内置数据类型.

表 2.9 MATLAB 内置数据类型

名 称	说 明
double	双精度浮点型
single	单精度浮点型
int8	有符号 8 位整数
uint8	无符号 8 位整数
int16	有符号 16 位整数
uint16	无符号 16 位整数
int32	有符号 32 位整数
uint32	无符号 32 位整数

2.12.2 模块支持的数据和数值信号类型

所有 Simulink 模块缺省状态下认为信号类型为双精度. 有些是逻辑(boolean)输入，有些则支持多种数据类型的输入. 表 2.10 给出了 Simulink 模块的数据类型.

表 2.10 Simulink 模块的数据类型

模块	说明
Abs	输入一个双精度类型的实或复信号，输出一双精度类型的实信号
Combinatorial Logic	如果逻辑模式可用，输入和输出数据类型是逻辑型；否则为双精度
Constant	输出一任何类型的实或复信号
Data Type Conversion	输入和输出任何实或复数据类型
Demux	接受混合类型的信号向量
Display	接受任何实或复数据类型
Dot Product	输入和输出任何实或复双精度类型值
Enable	相应的子系统的使能端口接受逻辑或双精度类型信号
From	输出与相连的 Goto 模块相对应的信号数据类型
From Workspace	输出与工作空间中的值相对应的类型
Gain	输入可以是实数或复数值信号，或除逻辑类型之外的任何数据类型向量
Goto	输入可以是任何类型
Ground	输出与其相连端口同样数据类型的 0 信号
Hit Crossing	输入双精度信号。如果逻辑模式是激活的，输出逻辑型；否则是双精度
Inport	输入端口接受任何类型实数或复数值信号。如果输入口是顶层输入端口，或者输入口与同一子系统的输出直接相连，输入信号向量的元素必须是相同类型
Integrator	积分器模块在其数据端口，接受和输出双精度类型的信号，其外部复位口接受双精度或逻辑类型数据
Logical Operator	如果逻辑模式是激活的，输入和输出逻辑类型的实信号；否则为双精度类型实信号
Manual Switch	接受任何类型的实数或复数值信号，所有输入必须具有相同的信号和数据类型
Math Function	输入和输出双精度类型的实数或复数值
MATLAB Function	输入和输出双精度类型的实数或复数值
Memory	输入任何类型的实数或复数值信号
Merge	输入和输出任何类型的实数或复数
Multiport Switch	多口开关模块的控制输入接受除逻辑类型之外的任何类型的实数值信号。其它输入接受任何类型的实数或复数值。所有输入必须是相同的数据和数值类型。模块的输出依赖于其输入
Mux	接受任何支持的 Simulink 数据类型，包括混合类型向量
Outport	接受任何 Simulink 数据类型作为输入。接受混合类型向量作为输入，如果其输出端口在子系统中，无需指定初始条件
Product	接受除逻辑类型之外的任何数据类型的实数和复数值信号
Relational Operator	接受任何支持的 Simulink 数据类型作为输入。每个输入必须类型相同。如果逻辑模式是激活的，输出逻辑类型；否则为双精度类型
Rounding Function	输入和输出双精度类型的实数或复数值
Scope	接受任何类型的实数或复数值信号

续表 2.10

模块	说明
Selector	输出所选输入信号的数据类型
Sum	接受任何支持的 Simulink 数据类型作为输入，所有输入必须类型相同；输出与输入类型一样
Switch	接受任何类型的实数或复数值信号作为开关输入(输入 1 和 3)，每个开关输入必须类型一样，模块输出信号与输入类型一致，阈值输入数据类型必须是逻辑型或双精度
Terminator	接受任何 Simulink 数据类型
To Workspace	接受任何 Simulink 数据类型作为输入
Trigger	相应子系统控制端口接受逻辑型或双精度类型信号
Trigonometric Function	输入和输出双精度类型的实数或复数值信号
Unit Delay	接受和输出任何类型的实数或复数值信号
Width	接受和输出任何类型的实数或复数值信号，包括混合类型信号向量
Zero-Order Hold	接受任何 Simulink 数据类型作为输入

2.12.3 指定模块参数的数据类型

当输入用户指定的数据类型的模块参数时，使用 type(value)语句来指定参数，其中 type 为数据类型的名称，value 为参数值.

下面列举几个定义参数值类型的例子：

single(2.5)　　指定一个单精度值 2.5

int8(3)　　指定一个 8 位整数值 3

int32(5+4i)　　指定一个实部和虚部为 32 位整数的复数值

2.12.4 产生指定数据类型的信号

采用下列方法可以导入指定类型的信号.

1) 经顶层输入或来自工作空间(From Workspace)模块，从 MATLAB 工作空间中调入所需类型的信号数据.

2) 在模型中创建一个常数模块，并将其参数设置成所需类型.

3) 使用数据类型转换(Data Type Conversion)模块来转换信号致所需的类型.

2.12.5 显示端口数据类型

要显示模型中的端口数据类型，从 Simulink 的 Format 菜单中，选择 Port Data Types 菜单项. 当改变框图中元素的数据类型后，Simulink 不更新端口数据类型显示. 要刷新显示，按 Ctrl+D 键.

2.12.6 数据类型传播

当开始仿真，并显示端口数据类型，或刷新端口数据类型显示时，Simulink 执行一个

称作数据类型传播的步骤.这一步骤包括,确定没有指定类型信号的数据类型,检验信号和输入端口的类型,保证它们不发生矛盾.如果类型矛盾出现,Simulink 显示一个错误对话框,指出数据类型发生矛盾的信号和端口.Simulink 同时还突出显示产生类型矛盾的信号路径.为解决类型矛盾,可以在模型中插入数据类型转换模块.

2.12.7 数据类型规则

下列这些规则有助于在建立模型时,解决类型安全问题,保证运行无错误.

1) 信号数据类型一般不影响参数数据类型;反之亦然.有一个重要的例外,就是常数模块.其输出数据类型是由其参数数据类型决定的.

2) 如果一个模块的输出是一个输入和一个参数的函数,并且输入和参数类型不一样,在运算输出前,Simulink 转换参数的类型为输入的类型.

3) 一般地,一个模块输出其输入出现的数据类型.重要的例外包括:常数模块和数据类型转换模块,它们的输出数据类型由其模块参数决定.

4) 虚拟模块在其输入端接受任何类型的信号.虚拟模块的例子包括 Mux 和 Demux 模块,以及无条件执行子系统.

5) 与非虚拟模块端口相连的信号向量元素必须数据类型一致.

6) 与非虚拟模块输入数据端口相连的所有信号的数据类型必须一致.

7) 控制端口(如 Enable 和 Trigger 端口)接受逻辑或双精度类型信号.

8) 求解器模块接受任何双精度信号.

9) 连接一个非双精度信号到模块,那么该模块不能进行过零点检测.

2.12.8 激活严格逻辑类型检测

缺省情况下,当检测到双精度信号与要求逻辑类型输入的模块相连时,Simulink 会检测到一个错误,但不通知该错误.这是为了保证由早期 Simulink 版本产生的,只支持双精度类型的模型的兼容性.可以激活严格逻辑类型检测,在仿真参数对话框中的诊断(Diagnostics)页中,不选 Relax boolean type checking 选项.

2.12.9 信号类型转换

只要检测到信号连接到了不接受的该信号数据类型的模块,Simulink 会发出错误通知.如果要生成该连接,必须将信号转换成模块接受的类型.转换可使用 Simulink 数据类型转换模块.

2.12.10 参数类型转换

一般地,在仿真时,在计算模块输出为输入信号和参数的函数时,Simulink 默认将参数数据类型转换为信号数据类型.下面是一些例外的情况:

1) 如果信号数据类型不能表达参数值,Simulink 停止仿真并通知错误信息.

2) 如果信号数据类型可以表示参数值,但需要降低精度,Simulink 会出现一个警告信息,并继续仿真.

2.13 处理复数信号

在缺省状态下，Simulink 信号值是实数. 尽管如此，模型可以产生和处理具有复数值的信号. 可以采取以下任何一种方法在模型中引入复数值信号.

1）由顶层端口从 MATLAB 工作空间向模型中装入复数值信号数据.

2）在模型中产生一个常数模块，并设置其值为复数.

3）产生实数信号分别对应于复数信号的实部和虚部，然后用实-虚复数转换(Real-Imag to Complex Conversion)模块将各部连成复数信号.

可以用接受复数的模块来处理复数信号，大多数 Simulink 模块接受复数信号作为输入.

2.14 保存模型

可以通过选择File 菜单下的Save 或者 Save As 命令保存模型. Simulink 是通过生成一个被称为模型文件(扩展名为.mdl)的有着特殊格式的文件来保存模型的，文件中存有模块图和模块的一些属性. 模型文件的格式将在后面章节中介绍.

如果是第一次保存一个模型，使用 Save 命令以提供一个文件名和它的路径. 模型文件名必须是以字母开头的且不能超过 31 个字母、数字、下划线组成的字符串.

如果保存一个已经保存过了的模型，用Save 命令以替换以前的内容，或者用 Save As 命令将模型存成一个新的名字或者位置.

Simulink 按以下步骤保存模型：

1）如果 mdl 文件已存在，将被命名为一个临时文件.

2）Simulink 执行所有模块预存储函数回调程序(PreSaveFcn callback routine)，然后执行模块图预存储函数回调程序.

3）Simulink 写模型文件至相同文件名，且扩展名为 mdl 的新文件.

4）Simulink 执行所有模块预存储函数回调程序，然后执行模块图预存储函数回调程序.

5）Simulink 删除临时文件.

在此过程中如果出现错误，Simulink 更名临时文件为原始模型文件名，写当前版本的模型到一个.err 扩展名的文件中，并出现错误信息. 即使在前面的步骤中出现了错误，Simulink 还是要执行第 2 步至第 4 步.

2.15 打印模块图

对于 Microsoft Windows 系统，可以通过选择File 菜单下的Print 命令打印模块图；或者对于所有平台，可以在 MATLAB 命令窗口中使用 Print 命令.

在 Microsoft Windows 系统中，Print 菜单项打印的是当前窗口中的模块图.

2.15.1 打印对话框

选择Print 菜单项时，将会出现打印对话框. 通过打印对话框可以有选择性地打印模型中的系统. 使用对话框，可以：

1) 只打印当前的系统；

2) 打印当前的系统及该模型层次结构中的所有上层系统；

3) 打印当前的系统及该模型层次结构中的所有下层系统，打印时可选择模板的内容和库模块；

4) 打印模型中的所有系统，打印时可选择模板的内容和库模块；

5) 打印每个图中的覆盖结构.

在打印对话框中支持选择性打印的部分在所有平台中都是相似的. 图 2.34 是在 Microsoft Windows 系统中的样子，图中只打印当前的系统：

当选择 Current system and below 或者 All systems 选项，两个复选框将被激活(enabled). 在图 2.35 中示出 All systems 选项被选择后的情况.

选择 Look Under Mask Dialog 复选框，当遇到模板子系统时将打印它的内容，或者打印当前模块的下层模块. 当打印所有系统时，上层系统被当作当前模块，所以 Simulink

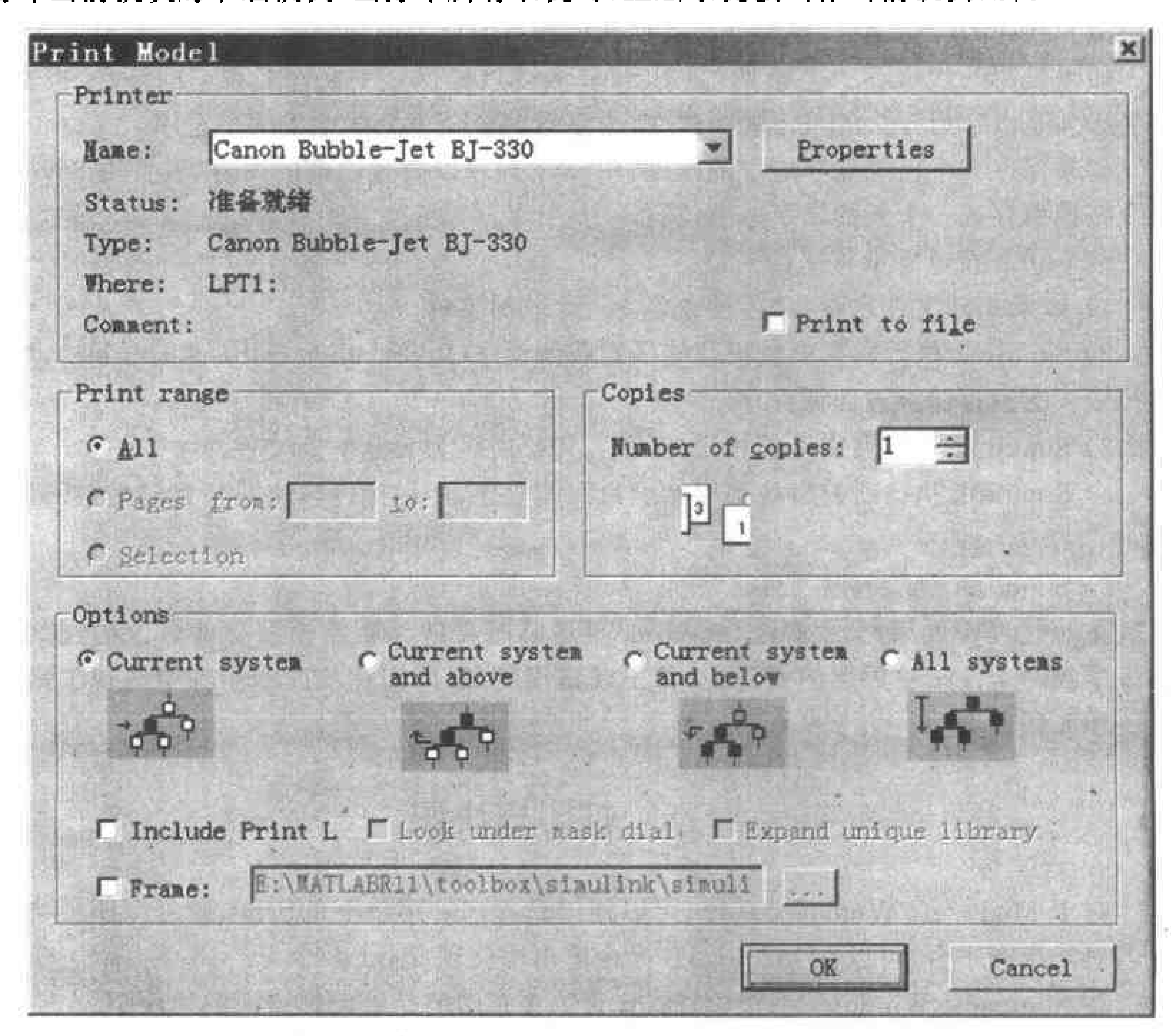


图 2.34 打印模型对话框

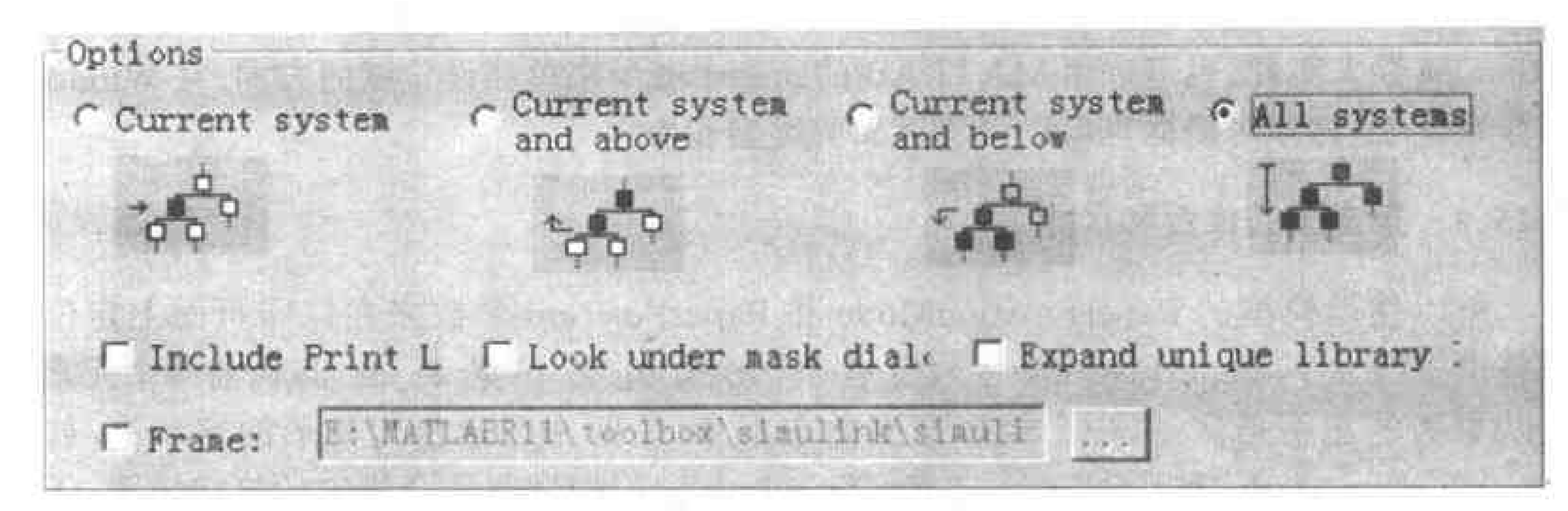


图 2.35 All systems 选项被选择

在遇到任何模板模块时都会深入其细节.

选择 Expand Unique Library Links 复选框,将打印库模块的内容,只要它们是系统.在模型中,只打印模块的一个拷贝,而不管该模块有多少个拷贝.

打印日志列出了打印的所有模块和系统,要打印日志,选择 Include Print Log 复选框.

选择 Frame 复选框,在每个图中打印一个小的模块框图.在附近的编辑框中输入命名模块框图的路径.使用 MATLAB 框图编辑器,可以自定义命名模块框图.在 MATLAB 命令行状态下,用 frameedit 或 help frameedit 命令可以了解其用途和方法.

2.15.2 打印命令

打印命令的格式如下:

print -ssys -device filename

sys 是要打印的系统的名字,系统名字必须以 s 选择标识符打头,并且是惟一的必要参数.在当前的任务中,sys 必须是打开的或曾经打开过的模型的名字,模型的名字必须是一个字符串.

device 指定设备类型.

filename 是要保存输出结果的 PostScript 文件名,如果文件名已经存在,它将被替换掉;如果文件名没有包含扩展名,将会自动地被加上合适的扩展名.

例如,下面的命令打印一个名字为 Demo 的系统:

print -sDemo

下面的命令打印当前系统中一个名为 Subx 的子系统的内容:

print -sSubx

下面的命令打印一个名为 Requisite Friction 的子系统的内容:

print (['-sRequisite Friction'])

下一个例子打印一个名为 Friction Model 的系统,该子系统的名字分为两行.第一个命令将一个换行符赋给一个变量;第二个命令打印系统:

cr=sprintf('\n');

print(['-sFriction' cr 'Model'])

2.15.3 指定纸张大小和方向

Simulink 允许指定打印模型图的纸张大小和方向.可以在任何操作平台上,使用

set_param 命令，通过分别设置模型的纸张类型(PaperType)和纸张方向(PaperOrientation)属性来实现．也可以用 MATLAB 的 orient 命令来单独设置纸的方向．在 Windows 操作系统中，打印对话框也可以设置页面的类型和方向．

2.15.4　指定图的位置和尺寸

可以使用模型的 PaperPositionMode 和 PaperPosition 参数来在打印页面上定位和调整模型图大小．PaperPosition 参数值是一个形式为[左 下 宽 高]的向量，前两个元素指定从页面的左下角测量，一个矩形范围的左下角．后两个元素指定矩形的宽和高．如果 PaperPositionMode 为手动(manual)时，Simulink 定位模型图适合指定的打印矩形．下面是一个设置打印参数的例子．

```
vdp
set_param('vdp' 'PaperType', 'B5')
set_param('vdp', 'PaperOrientation', 'landscape')
set_param('vdp', 'PaperPositionMode', 'manual')
set_param('vdp', 'PaperPosition', [0.5 0.5 4 4])
print -svdp
```

在 B5 纸的左下角，横向打印 dvp 模型的模块图．

如果 PaperPositionMode 为自动(auto)时，Simulink 在打印页面上居中模型图，必要时调整图，以适应页面的大小．

2.16　模块浏览器

使用模型浏览器可以：

1) 浏览模型层次结构；
2) 在模型中直接打开系统；
3) 确定模型中包含的模块．

浏览操作在不同的操作系统中不一样．在 Windows 平台下，从 Simulink 的 View 菜

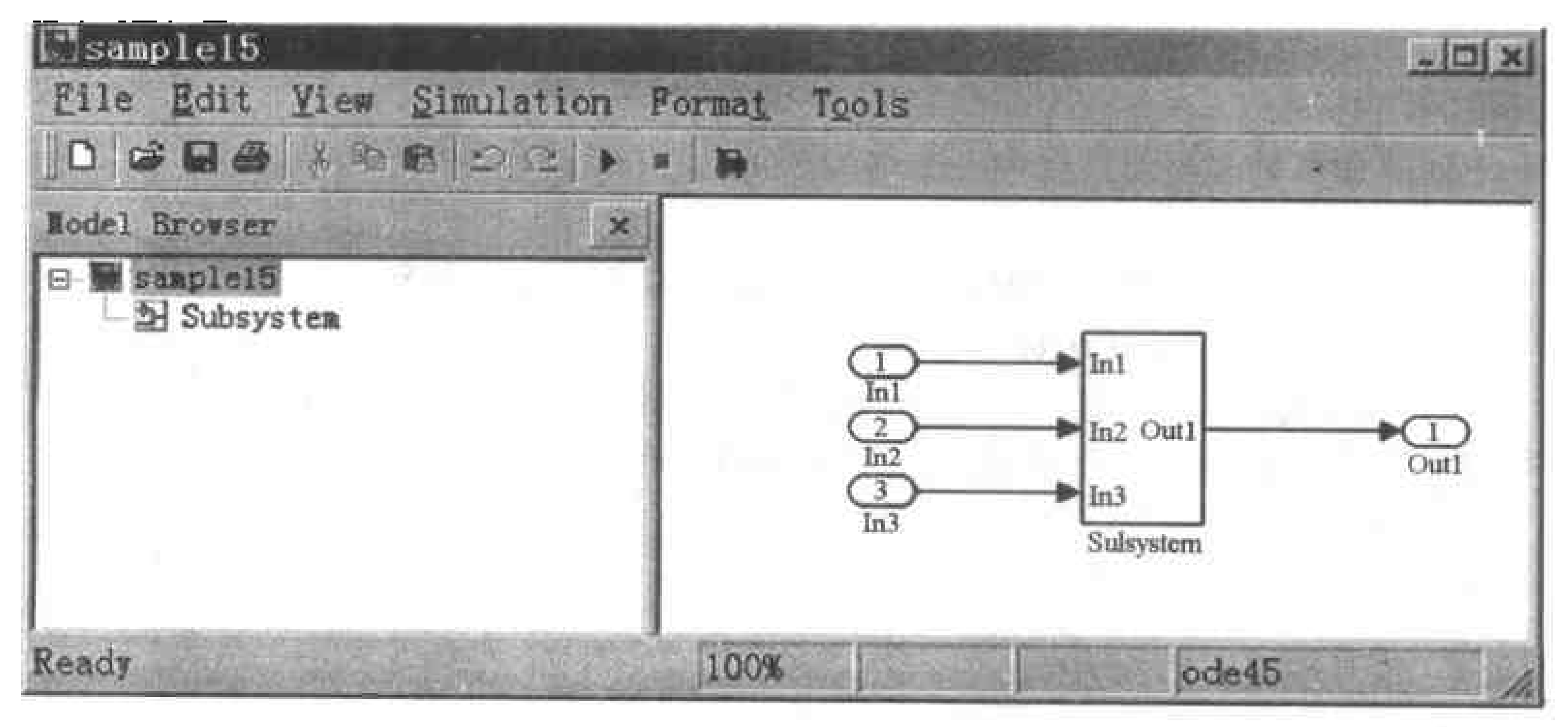


图 2.36　模块浏览器窗口

单中选择Model Browser 菜单项，就可显示模型浏览窗口. 左边的框子中显示一个树结构，右边框内显示模块图. 图 2.36 是浏览器窗口的样子。

2.17　跟踪模型版本

一个 Simulink 模型在其开发过程中可经历多个版本，Simulink 可以通过产生和存储版本控制信息来帮助跟踪不同的版本. 版本控制信息包括版本数、创建人、模型的最近修改以及改变的历史记录. 以下一些 Simulink 功能允许使用和管理版本控制信息.

1）使用 Simulink 模型参数对话框，可以编辑存储在模型中的版本控制信息，并且可以选择各种版本的控制选项.

2）使用 Simulink 模型信息（Model Info）模块，可以显示版本控制信息，包括由外部版本控制系统维护的，作为模型图的注释模块.

3）使用 Simulink 版本控制参数，可以从 MATLAB 命令行或 M-文件来直接访问版本控制信息.

2.17.1　指定当前用户

当一个用户创建或改变一个模型，为了版本控制，Simulink 在模型中记录下用户的名字. Simulink 假定用户名至少由下列环境变量中的一个指定：USER，USERNAME，LOGIN，或 LOGNAME. 如果系统没有定义这些变量，Simulink 不改变模型中的用户名. UNIX 系统通常定义 USER 环境变量，并设置其值为登录系统所使用的名字. 因此，使用 UNIX 系统，没有必要执行任何操作来识别当前用户. Windows 系统中，可以定义一些或没有 Simulink 所需的“用户名”的环境变量，这要依据安装 Windows 系统的版本，以及是否是单机或连网运行. 可以使用 MATLAB 的 getenv 命令来确定哪个环境变量定义了. 例如，在 MATLAB 命令行中输入：getenv ('user')来确定在当前的 Windows 系统中是否存在 USER 环境变量. 如果没有，必须亲自设定. 在 Windows 95 和 98 下，在系统的 autoexec.bat 文件中输入命令：set user=myname，其中，myname 就是用来识别在模型中的用户名字，然后重新启动计算机. autoexec.bat 文件一般在系统盘的根目录中可找到.

在 Windows NT 上，使用系统属性对话框的环境变量面板来设置 USER 环境变量，如图 2.37 所示.

2.17.2　模型属性对话框

模型属性对话框可用来编辑一些版本控制参数，并设置相关的选项. 从 Simulink 的 File 菜单中选择 Model Parameters，即显示该对话框，如图 2.38 所示.

2.17.2.1　模型属性板

模型属性板（Model Properties），如图 2.38 所示，可以编辑以下版本控制参数：

1）创建者（Creator）：创建该模型者的名字，Simulink 设置该属性为创建模型时 USER环境变量的值. 编辑该字段，改变其值.

2）创建于（Created）：模型创建的时间和日期.

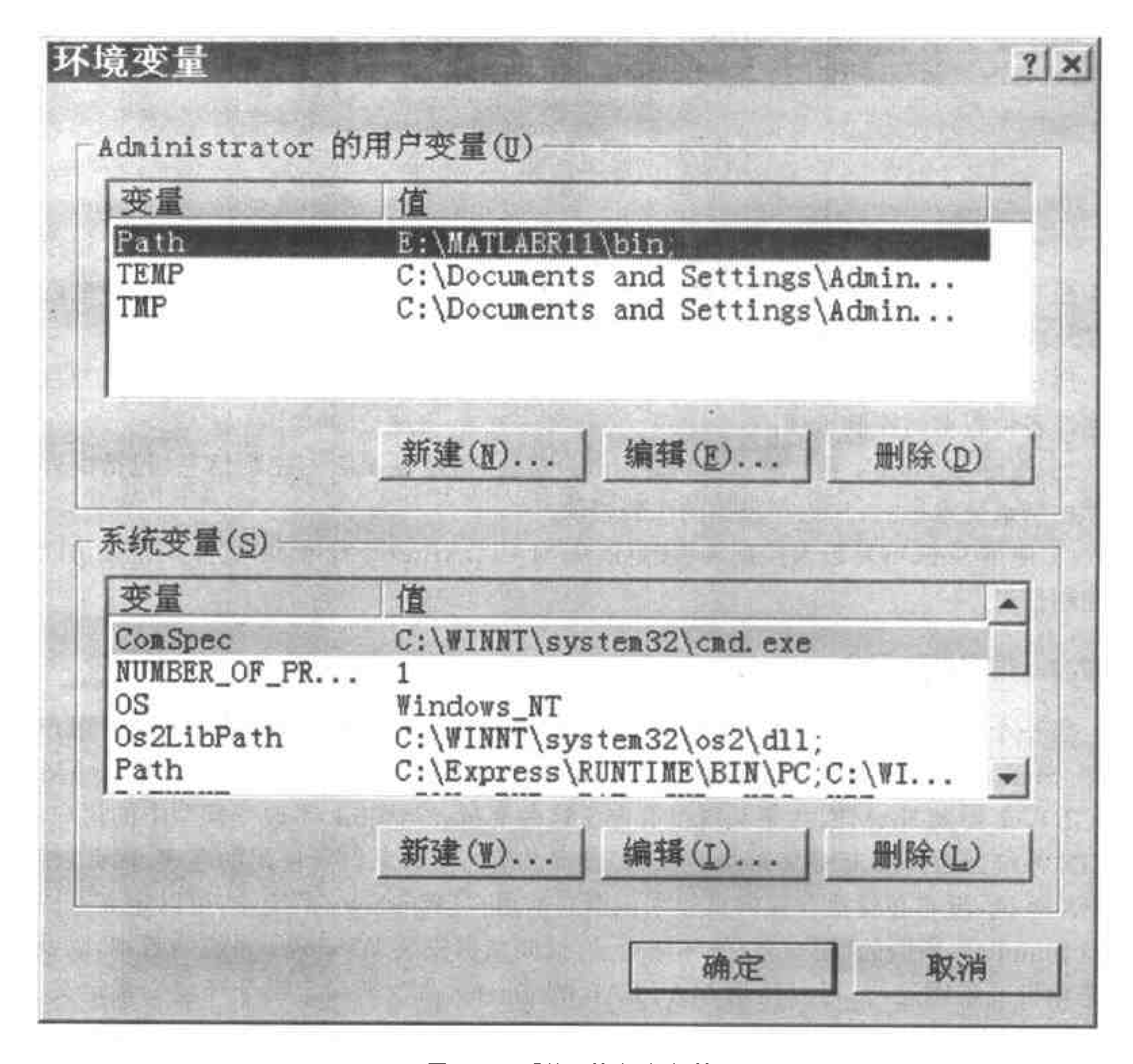


图 2.37 系统环境变量对话框

3）模型描述(Model description)：对模型的解释和说明.

2.17.2.2 选项板

选项板可以选择一个配置管理人，并指定版本控制信息格式，如图 2.39 所示.

1）配置管理者(Configuration manager)：外部配置管理者用于管理该模型，选择该选项，可以包括来自模型信息注释模块中的配置管理者. 在 MATLAB 根目录下的/toolbox/local 目录中的文件 cmopts.m 指定了模型的缺省配置管理者. 缺省值“default”没有配置管理者. 用户可以编辑该文件来指定另外的选择.

2）模型版本格式(Model version format)：格式用于在模型参数板和模型信息模块中显示模型版本数. 该参数的值可以是任何文本字符串，文本字符串可以包含%<AutoIncrement：#>标记，其中#是一个整数. 当显示模型版本数时，Simulink 用#代替标记. 如：1. %<AutoIncrement：3>，显示为 1.3. 当保存模型时，Simulink 将#增加 1. 例如，当保存模型后，1. %<AutoIncrement：3>就变为了 1. %<AutoIncrement：4>.

3）修改者格式("Modified by" format)：格式用于在历史记录板上显示修改者的值，

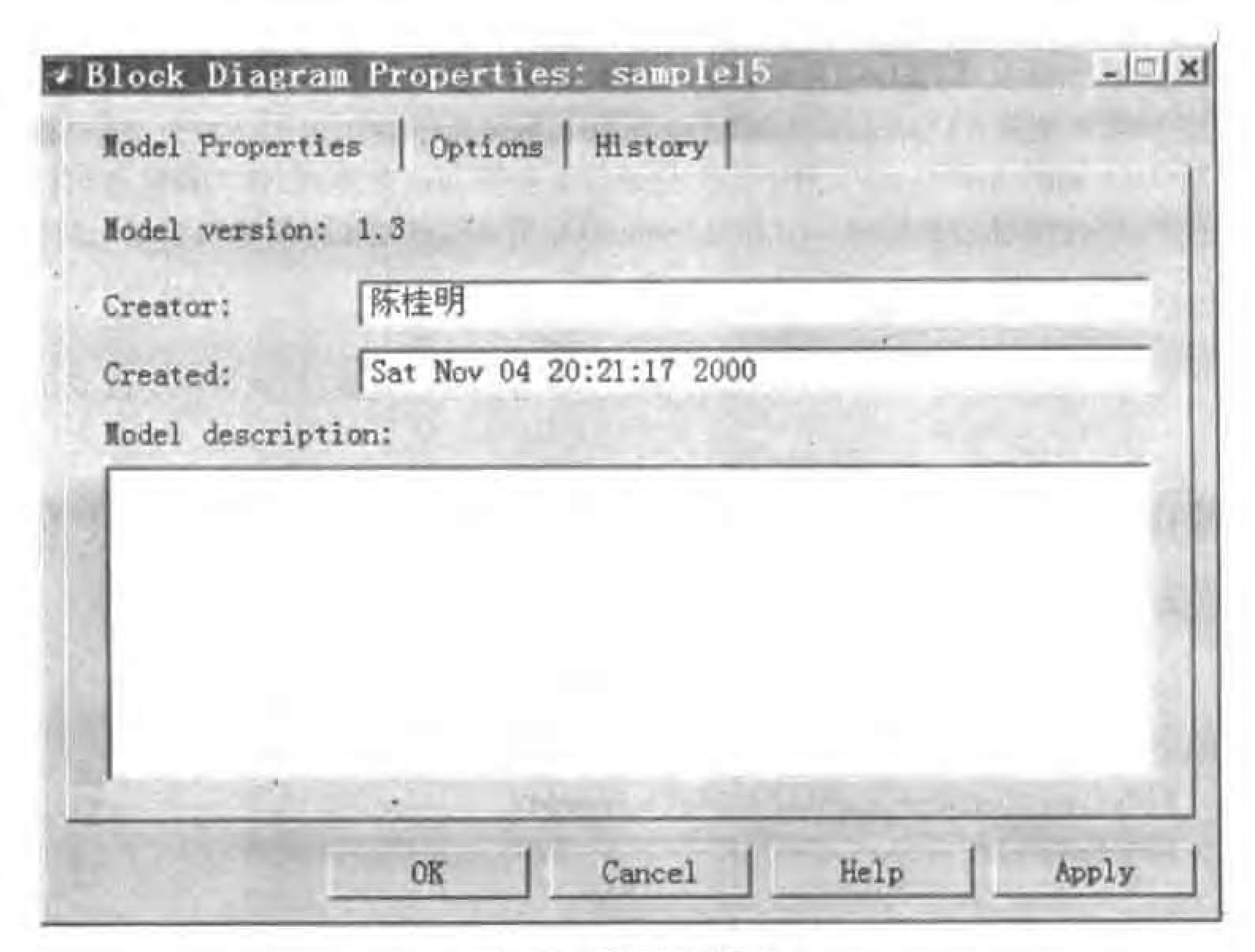


图 2.38 模型属性对话框

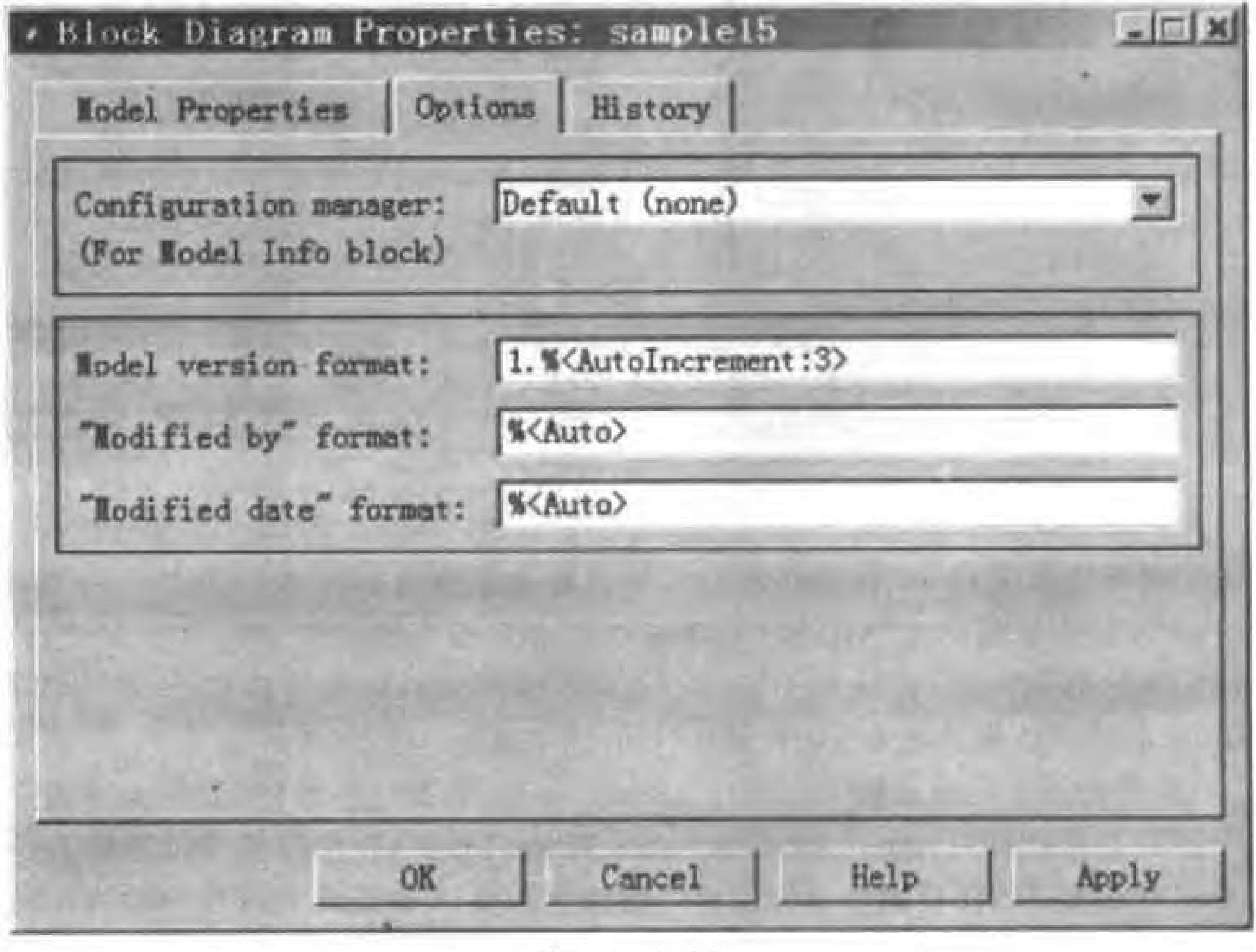


图 2.39 选项板

用于历史记录标志和模型信息模块. 该字段的值可以是任意字符串,字符串可包含%〈Auto〉标记. Simulink 用 USER 环境变量的当前值来替代该标记.

4) 修改日期格式("Modified date" format):格式用于在历史记录板上显示"最后修改日期(Last modified date)",用于历史记录标志和模型信息模块. 该字段的值可以是任意字符串,字符串可包含%〈Auto〉标记. Simulink 用当前日期和时间替换该标记的值.

2.17.2.3　历史记录板

历史记录板如图 2.40 所示. 历史记录板使得用户可以观察、编辑该模型改变历史记录.

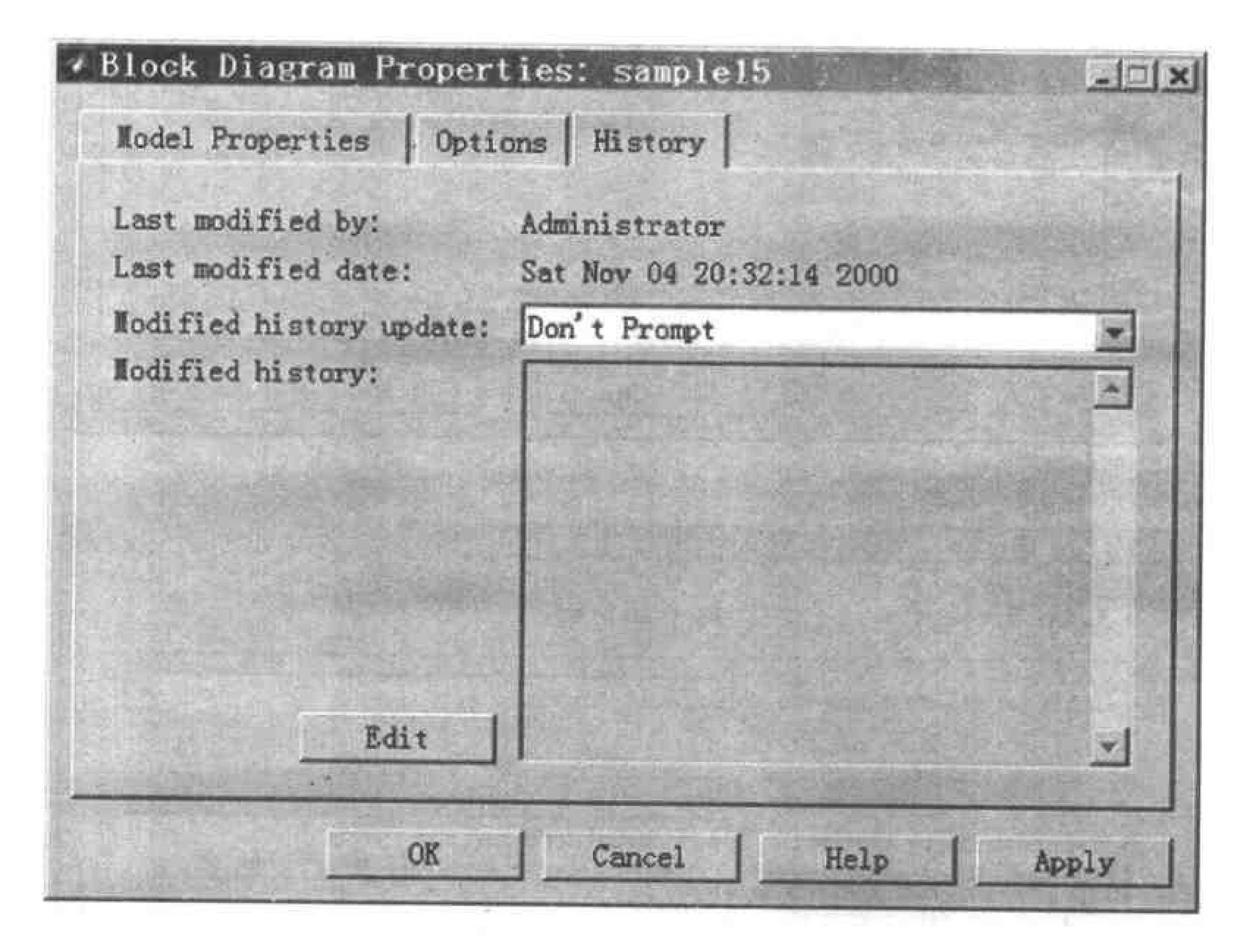


图 2.40　历史记录板

1) 最后修改者(Last modified by):最后修改该模型的人名. 当保存一个模型时,Simulink 将该参数值设置为 USER 环境变量值. 该字段不能编辑.

2) 最后修改日期(Last modified date):该模型最后修改的日期,当模型保存时,Simulink 将该参数值设置为系统日期和时间. 该字段也不能编辑.

3) 修改的历史记录更新(Modified history update):指定在保存模型时是否提示一个用户说明. 如果选择"Prompt for Comments When Save",在保存模型时将提示一个说明. 用户可以使用该注释说明对模型所作的任何修改. Simulink 保存该参数的先前值于模型的改变历史记录中.

4) 修改的历史记录(Modified history):模型修改的历史记录,当用户修改模型时,

Simulink 将从用户输入的注释中编译历史记录. 用户任何时间都可以按其附近的 Edit 按钮来编辑历史记录.

2.17.3　创建模型改变历史记录

Simulink 允许用户创建和保存模型改变的记录于模型中，当保存模型时，Simulink 将从用户输入的注释中自动编译历史记录.

2.17.3.1　保存改变(Logging Changes)

要改变历史记录，必须在模型属性对话框中的历史记录板中，选择"Prompt for Comments When Save"，下次保存模型时，就会显示如图 2.41 所示的保存改变(Log Change)对话框.

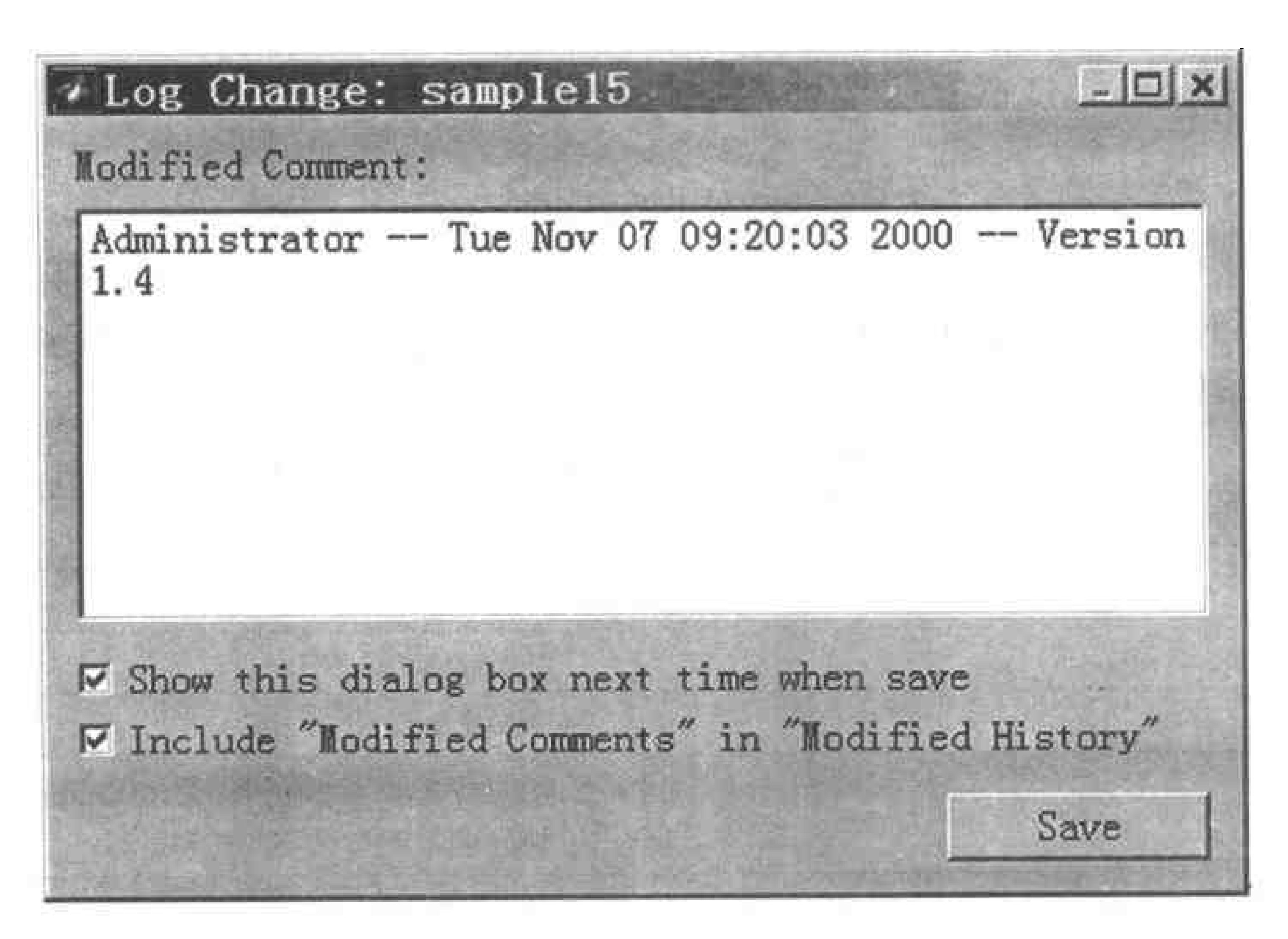


图 2.41　保存改变对话框

如果想在模型改变历史记录中增加项目，在修改注释(Modified Comments)编辑框中输入项目，并按保存(Save)按钮. 如果不想输入项目，不选 Include "Modified Contents" in "Modified History"选项的复选框；如果想不再改变记录，不选 Show this dialog box next time when save 选项的复选框.

2.17.3.2　编辑改变历史记录

要编辑模型改变历史记录，按模型属性对话框的历史记录板中的编辑(Edit)按钮，Simulink 会显示如图 2.42 所示的修改历史记录(Modification History)对话框.

在该编辑框中显示的历史记录，按应用(Apply)或 OK 按钮保存改变.

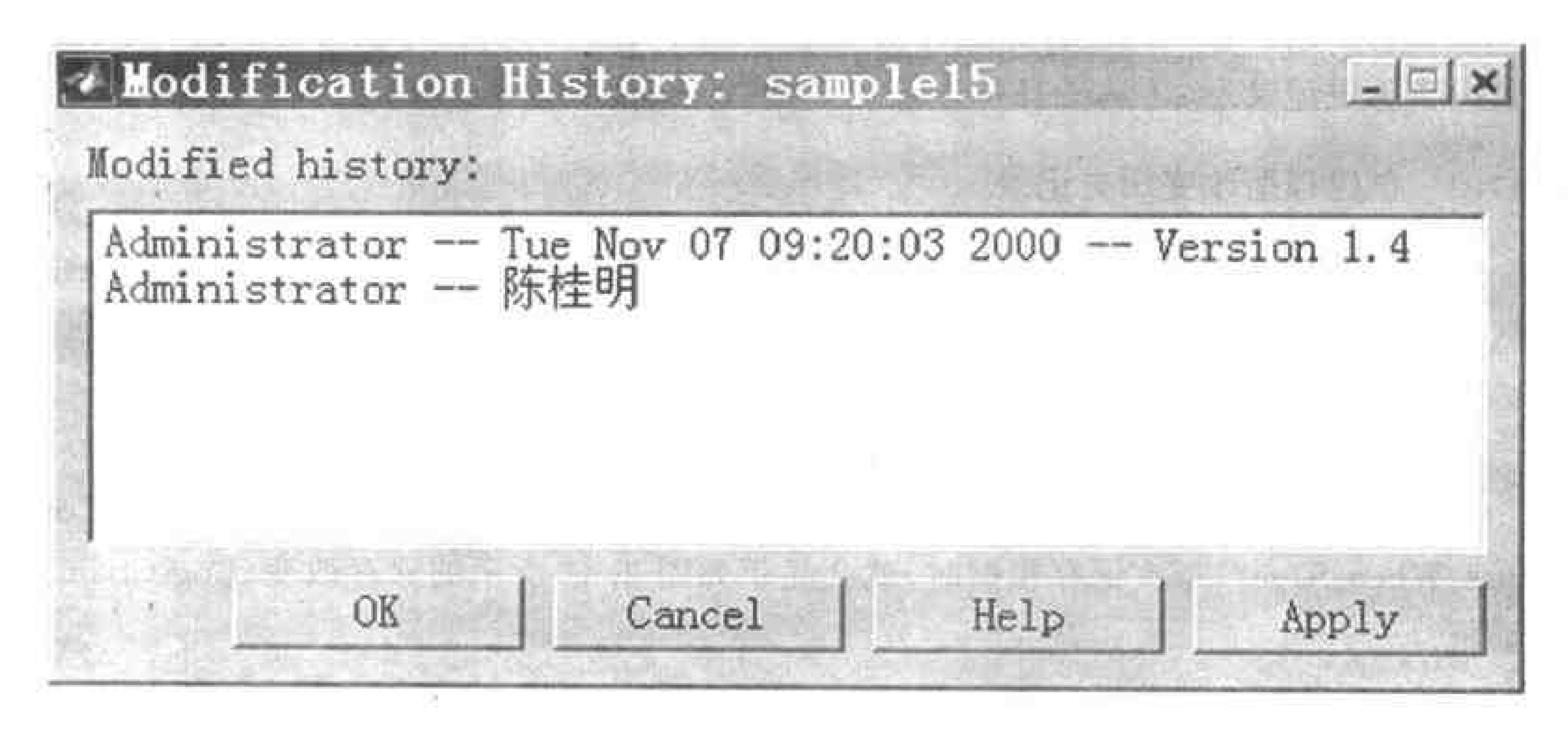


图 2.42　修改历史记录对话框

2.17.4　版本控制属性(Version Control Properties)

Simulink 在模型中保存版本控制属性作为模型参数,用户可以用 get_param 命令,从 MATLAB 命令行或 M-文件来访问该信息.表 2.11 列出了 Simulink 用来保存版本控制信息的模型参数.

表 2.11　Simulink 用来保存版本控制信息的模型参数

属　性	说　明
Created	创建日期
Creator	创建该模型者的名字
ModifiedBy	最后修改者的名字
ModifiedByFormat	最后修改者的名字格式,可以为字符串,字符串可包含 %(Auto)标记,Simulink 使用当前 USER 环境变量来代替该标记
ModifiedDate	修改日期
ModifiedDateFormat	修改日期格式,可以为字符串,字符串可包含 %(Auto)标记,当保存模型时,Simulink 使用当前日期和时间代替该标记
ModifiedComment	最后修改模型者输入的注释
ModifiedHistory	模型改变的历史记录
ModelVersion	版本数
ModelVersionFormat	版本数格式,可以为字符串,字符串可包含%(AutoIncrement:#)标记,其中#为整数,显示版本数时用#的值代替标记,当保存模型时#值加 1
Description	模型的描述或介绍
LastModificationDate	最后修改日期

2.18 模型构造命令

表 2.12 给出了本节将要介绍的命令所执行的任务.

表 2.12 创建模型的命令

命令	任务
new_system	创建新的 Simulink 系统
open_system	打开已存在的系统
close_system, bdclose	关闭某一系统的窗口
save_system	保存 Simulink 系统
find_system	查找 Simulink 系统、模块、线或注释
add_block	向系统加入新的模块
delete_block	从系统中删除模块
replace_block	在系统中替换模块
add_line	向系统上加入连线
delete_line	从系统删除连线
get_param	获取参数值
set_param	设置参数值
gcb	获取当前模块的路径
gcs	获取当前系统的路径
gcbh	获取当前模块的句柄
bdroot	获取顶层系统的名字
simulink	打开 Simulink 模块库

2.18.1 指定 Simulink 对象的路径

本节中描述的大多数命令需要标识一个 Simulink 系统或者模块. 通过指定它们的路径标识系统和模块：

1）要标识一个系统，指定系统的名字，也就是包含系统描述的文件名字，不要带 mdl 扩展名.

2）标识一个子系统，指定该子系统所处的系统及子系统层次：

所在系统/子系统 1/.../要指定的子系统

3）要标识一个模块，指定包含该模块的系统的路径及模块的名字：

所在系统/子系统 1/.../子系统/模块名字

如果模块的名字包括回车或者换行符，指定该模块的名字为一个字符串，并且使用 sprintf('\n')作为换行符. 例如，下面这些命令行将换行符赋值给 cr 变量，然后取得 Signal Generator 模块的 Amplitude 参数的值：

```
cr=sprintf('\n');
get_param(['untitled/Signal', cr , 'Generator'], 'Amplitude')
```

ans＝

　1

如果模块的名字包含有斜杠线(/)，当指定模块的名字时应该输入两个斜杠线. 例如，要取得 mymodel 系统中名为 Signal/Noise 模块的 Location 参数的值，命令输入的方法如下：

get_param('mymodel/Signal//Noise', 'Location')

2.18.2　命令 add_block

功能：向 Simulink 系统加模块.

语法：add_block('source', 'destination')

　　add_block('source', 'destination', 'parameter1', value1, ...)

说明：

add_block('source', 'destination')：拷贝一个包含有完整路径名称 'source' 的模块到系统中，并指明新的模块的完整路径名'destination'. 新的模块的参数与原来的模块的参数相同. 可以用'build-in'作为所有 Simulink 内建(Simulink 模块库中可用的并且不是模板的模块)模块的源系统名字.

add_block('source', 'destination', 'parameter1', value1, ...)：创建如上面所说的一个拷贝，但所列的参数具有被指定的值. 任何另外指定的参数必须参数和值成对地出现.

例 2.1　下面的命令从 simulink 系统的 Sinks 子系统中拷贝 Scope 模块到 engine 系统的 timing 子系统中，名字换成 Scope-1：

add_block('simulink/Sinks/Scope', 'engine/timing/Scope-1')

下面的命令在 F14 系统中创建一个名为 controller 的新的子系统：

add_block('built-in/SubSystem', 'F14/controller')

下面的命令拷贝内建的 Gain 模块到 mymodel 系统，新的模块的名字为 Volume，并且指定它的 Gain 参数值为 4：

add_block('built-in/Gain', 'mymodel/Volume', 'Gain', '4')

参见：delete_block，set_param 命令

2.18.3　命令 add_line

功能：加连线到 Simulink 系统.

语法：h＝add_line('systemname', 'outport', 'inport')

　　h＝add_line('systemname', points)

说明：

add_line 命令在指定的系统中增加一条连线，并返回新连线句柄 h. 这条连线有两种定义方法：

1）指明要被连线连接的端口的名字.

2）指定定义连线线段的端点的位置.

add_line('systemname', 'outport', 'inport')：在系统中增加一条从指定的模块输出

端口'outport'到指定的模块输入端口'inport'的连线.'outport'和'inport'是由模块名字和端口标识以"模块/端口"的形式组成的字符串.大多数模块端口是通过从顶层到底层、从左到右的顺序编号加以标识的,如:'Gain/1'或者'Sum/2'等.而 Enable(激活)、Trigger(触发)和 State(状态)端口是通过名字加以标识,如:'subsystem_name/Enable','subsystem_name/Trigger',或者'Integrator/State'等.

add_line('systemname', points);增加线段到一个系统.数组 points 的每一行指定线段端点的 x 和 y 坐标.坐标原点是窗口的左上角.信号从第一行定义的点传到最后一行定义的点.如果新的连线的起点靠近一个已存在的模块的输出或者一条连线,它们之间将会相连.同样,如果连线的末端靠近一个已经存在的输入,它们之间也将会连接起来.

例 2.2 下面的命令在 mymodel 系统中增加一条连线以连接 Sine Wave 模块的输出到 Mux 模块的第一个输入:

add_line('mymodel', 'Sine Wave/1', 'Mux/1')

下面的命令在 mymodel 中增加一条连线,该连线分成两段,第一段从(20,55)到(40,10),第二段从(40,10)到(60,60):

add_line('mymodel', [20 55; 40 10; 60 60])

参见:delete_line 命令

2.18.4 命令 bdclose

功能:无条件地关闭任何或所有 Simulink 系统的窗口.

语法:bdclose

bdclose('systemname')

bdclose('all')

说明:

bdclose;不带参数的命令,无条件且不经确认地关闭当前系统的窗口.该系统从最后一次保存到关闭时所做的修改都将丢失.

bdclose('systemname');关闭指定系统的窗口.

bdclose('all');关闭所有系统的窗口.

例 2.3 下面的命令关闭 sample15 系统:

bdclose('sample15')

参见:close_system,new_system,open_system,save_system 命令

2.18.5 命令 bdroot

功能:返回顶层 Simulink 系统的名字.

语法:bdroot

bdroot('objectname')

说明:

bdroot;不带参数的命令,返回顶层系统的名字.

bdroot('objectname');返回包含指定对象名字的顶层系统的名字,其中'objectname'是系统或者模块路径的名字.

例 2.4 下面的命令返回包含有当前模块的顶层系统的名字：

bdroot(gcb)

ans =

sample15

参见：find system,gcb 命令

2.18.6 命令 close_system

功能：关闭 Simulink 系统的窗口或者模块的对话框.

语法：close_system

close_system('systemname')

close_system('systemname', saveflag)

close_system('systemname', 'newname')

close system('blockname')

说明：

close_system：不带参数的命令，关闭当前系统或者子系统的窗口. 如果当前系统是顶层系统并且它被修改过，那么该命令在将系统从内存清除之前询问是否将修改过的系统保存到文件. 当前系统由 gcs 命令定义.

close_system('systemname')：关闭指定的系统或者子系统窗口.

close_system('systemname', saveflag)：关闭指定的顶层系统窗口并且将它从内存清除掉：

如果 saveflag 是 0，系统不保存.

如果 saveflag 是 1，系统用当前的名字保存.

close_system('systemname', 'newname')：用指定的新的名字 newname 保存指定的顶层系统 systemname，然后关闭系统.

close_system('blockname')：关闭与指定的模块关联的对话框，或者如果定义了 CloseFcn 回调函数，就调用模块的 CloseFcn 函数，其中'blockname'是完整的模块路径名称. 任何另外的参数都将忽略.

例 2.5 下面的命令关闭当前的系统：

close_system

下面的命令关闭 sample15 系统：

close_system('sample15')

下面的命令用它当前的名称保存 sample15 系统，然后关闭它：

close_system('sample15', 1)

下面的命令用 testsys 的名字保存 mymodel 系统，然后关闭它：

close_system('mymodel', 'testsys')

下面的命令关闭 engine 系统的 Combustion 子系统中的 Unit Delay 模块的对话框：

close_system('engine/Combustion/Unit Delay')

参见：bdclose,gcs,new_system,open_system,save_system 命令

2.18.7 命令 delete_block

功能：从 Simulink 的系统中删除模块.

语法：delete_block('blockname')

说明：

delete_block('blockname')：从系统中删除指定的模块，其中'blockname'是完整的模块路径名.

例 2.6 下面的命令从 vdp 系统中删除 Out1 模块：

```
delete_block('vdp/Out1')
```

参见：add_block 命令

2.18.8 命令 delete_line

功能：从 Simulink 系统中删除连线.

语法：delete_line('systemname', 'outport', 'inport')

　　delete_line('systemname',[x y])

说明：

delete_line('systemname', 'outport', 'inport')：删除从指定的模块输出端口'outport'到指定的模块输入端口'inport'的连线.'outport'和'inport'是由模块名字和端口标识以'block/port'的形式组成的字符串.大多数模块端口是通过从顶层到底层、从左到右的顺序编号加以标识的，如'Gain/1'或者'Sum/2'. Enable(激活)、Trigger(触发)和 State(状态)端口是通过名字加以标识，如'subsystem_name/Enable'、'subsystem_name/Trigger'或者'Integrator/State'.

delete_line('systemname',[x y])：删除系统中包含有指定点(x, y)的连线，如果这样的连线存在的话.

例 2.7 下面的命令删除 mymodel 系统中连接 Sum 模块与 Mux 模块第二个输入的连线：

```
delete_line('mymodel', 'Sum/1', 'Mux/2')
```

参见：add_line 命令

2.18.9 命令 find_system

功能：查找 Simulink 系统、模块、连线和注释.

语法：find_system (sys,'constraint',cv,'parameter1', value1, 'parameter2', value2, ...)

说明：

find_system (sys, 'constraint',cv,'parameter1', value1, 'parameter2', value2, ...)：搜索由 sys 指定的系统或子系统，用'constraint'指定的约束，并返回句柄或路径至由参数与值指定的对象. sys 可以是路径名(或路径名数组)、句柄(或句柄向量)，也可以省略. 如果 sys 是路径名或路径名数组，则返回查找对象的路径名数组；如果 sys 是句柄或句柄向量，则返回查找对象的句柄向量. 如果没有 sys 参数，搜索所有打开系统.

参数名字忽略大小写，字符串形式的参数值要区分大小写. 与对话框的条目相关联的任何参数具有字符形式的值. 可以指定表 2.13 中的任何搜索约束.

表 2.13 创建模型的命令

名　字	值类型	说　明
'SearchDepth'	标量(scalar)	限制指定的深度(0 仅搜索打开系统，1 为顶层系统的子系统或模块，2 为顶层系统及其子层等)，缺省为所有层
'LookUnderMasks'	'on' | 'off'	'on'，搜索扩展的模板系统，缺省为 'off'
'FollowLinks'	'on' | 'off'	如果为'on'，搜索相连的模块库，缺省为 'off'
'FindAll'	'on' | 'off'	如果为'on'，搜索扩展到系统中的连线和注释，缺省为 'off'

如果'constraint'省略，则使用缺省的约束值.

例 2.8 下面的命令返回一个包含所有打开的系统和模块的名字的单元数组：

```
find_system
ans =
        'sample15'
        'sample15/In1'
        'sample15/In2'
        'sample15/In3'
        'sample15/Subsystem'
        'sample15/Subsystem/In1'
        'sample15/Subsystem/In2'
        'sample15/Subsystem/In3'
        'sample15/Subsystem/Gain'
        'sample15/Subsystem/Sum'
        'sample15/Subsystem/Out1'
        'sample15/Out1'
```

下面的命令返回所有打开的模块图的名字：

```
open_bd=find_system('Type', 'block_diagram')
open_bd =
        'sample15'
```

下面的命令返回 clutch 系统的 Unlocked 子系统中的所有 Goto 模块：

```
find_system('clutch/Unlocked', 'SearchDepth', 1, 'BlockType', 'Goto')
ans =
        'clutch/Unlocked/Goto'
        'clutch/Unlocked/Goto1'
```

下面这些命令返回 vdp 系统中所有 Gain 参数值为 1 的 Gain 模块：

```
gb=find_system('vdp', 'BlockType', 'Gain');
find_system(gb, 'Gain', '1')
ans =
```

'vdp/Mu'

上面的命令等价于下面的命令：

find_system('vdp', 'BlockType', 'Gain', 'Gain', '1')

ans =

'vdp/Mu'

参见：get_param，set_param 命令

2.18.10 命令 gcb

功能：取得当前 Simulink 模块的完整路径名.

语法：gcb

gcb('systemname')

说明：

gcb：返回当前系统中当前模块的完整路径名.

gcb('systemname')：返回指定系统中当前模块的完整路径名.

当前模块是下面所列情况中的一种：

1）编辑时，当前模块是最近点击的模块.

2）在运行一个包含有 S-Function 模块的系统的仿真时，当前模块是当前正在执行其相应的 MATLAB 函数的 S-Function 模块.

3）在回调时，当前模块是正在执行回调程序的模块.

4）在计算 MaskInitialiazation 字符串时，当前模块是那个正在计算它的模板的模块.

例 2.9 下面的命令返回最近被选取的模块的路径：

gcb

ans =

clutch/Friction Mode Logic/Requisite Friction/Inertia

Ratio

下面的命令取得当前模块的 Gain 参数的值：

get_param(gcb, 'Gain')

ans =

Iv/(Iv+Ie)

参见：gcbh，gcs 命令

2.18.11 命令 gcbh

功能：取得当前 Simulink 模块的句柄.

语法：gcbh

说明：

gcbh：返回当前系统中当前模块的句柄.

可以使用该命令标识没有父系统的模块或者给出它的地址. 这一命令对于模块集的作者非常有用.

例 2.10 下面的命令返回最近被选取的模块的句柄：

gcbh

ans =

    114.0001

参见:gcb 命令

2.18.12 命令 gcs

功能:取得当前 Simulink 系统的完整路径名.

语法:gcs

说明:

gcs:返回当前系统的完整路径名.

当前系统是:

1) 在编辑时,当前系统是最近在其中点击过的系统或子系统.

2) 在运行一个包含有 S-Function 模块的系统的仿真时,当前系统是包含当前正在计算的 S-Function 模块的系统或者子系统.

3) 在回调时,当前系统是包含正在执行其回调程序模块的系统.

4) 在计算 MaskInitialiazation 字符串时,当前系统是包含正在计算其模板的模块的系统.

例 2.11 下面的例子返回包含最近被选取的模块的系统的路径:

gcs

ans =

clutch/Friction Mode Logic/Requisite Friction

参见:gcb 命令

2.18.13 命令 get_param

功能:取得 Simulink 的系统或者模块的参数的值.

语法:get_param('objectname', 'parameter')

    get_param({objects}, 'parameter')

    get_param(handle, 'parameter')

    get_param('objectname', 'ObjectParameter')

    get_param('objectname', 'DiaglogParameter')

说明:

get_param('objectname', 'parameter'):返回指定参数的值,其中'objectname'是系统或者模块的路径名.参数名忽略大小写.

get_param({objects}, 'parameter'):接受一个指定完整路径的单元数组,使得用户能够取得在单元数组中指定的所有对象所共有的参数的值.

get_param(handle, 'parameter'):返回句柄为 handle 的对象的指定参数.

get_param('objectname', 'ObjectParameter'):返回描述 objectname 参数的结构.返回结构的每个字段与各自的参数相对应,并具有参数名称.例如,名字字段与对象的名字参数相对应.每个参数字段包含三个字段:名称、类型和属性,分别指定参数的名称、数据

类型和属性.

get_param('objectname', 'DiaglogParameter');返回包含指定模块对话参数名的单元数组.

例 2.12 下面的命令返回 clutch 系统的 Requisite Friction 子系统中的 Inertia Ratio 模块的 Gain 参数的值:

```
cr=sprintf('\n');
get_param(['clutch/Friction Mode Logic/Requisite Friction/Inertia',cr,'Ratio'],'Gain')
ans =
Iv/(Iv+Ie)
```

下面这些命令显示当前系统中所有模块的模块类型:

```
gcs
ans =
clutch/Friction Mode Logic/Requisite Friction
blks=find_system(gcs, 'Type', 'block');
listblks=get_param(blks, 'BlockType')
listblks =
    'SubSystem'
    'Inport'
    'Gain'
    'Gain'
    'Sum'
    'Sum'
    'Gain'
    'From'
    'Outport'
```

下面命令返回当前所选模块的名字:

```
gcb
ans =
clutch/Friction Mode Logic/Requisite Friction/Inertia
Ratio
get_param(gcb, 'Name')
ans =
Inertia
Ratio
```

下面命令获取当前所模块的名字参数的属性:

```
p = get_param(gcb, 'ObjectParameters');
a = p.Name.Attributes
a =
```

'read-write' 'always-save'

下面的命令获取 Sine Wave 模块的对话参数：

```
p = get_param('untitled/Sine Wave', 'DialogParameters')
p =
     Amplitude: [1x1 struct]
     Frequency: [1x1 struct]
         Phase: [1x1 struct]
    SampleTime: [1x1 struct]
```

参见：find_system，set_param 命令

2.18.14 命令 new_system

功能：创建新的空 Simulink 系统.

语法：new_system('systemname')

说明：

new_system('systemname')：用指定的名字创建一个新的空的系统. 如果'systemname'指定的是路径，那么将会在指定的路径下的系统中创建一个新子系统. new_system 不打开系统的窗口.

例 2.13 下面的命令创建一个名为'mysystem'的新的系统：

```
new_system('mysystem')
```

下面的命令在 vdp 系统中创建一个名为'mysystem'的新的子系统：

```
new_system('vdp/mysystem')
```

参见：close_system，open_system，save_system 命令

2.18.15 命令 open_system

功能：打开一个 Simulink 系统窗口或者模块的对话框.

语法：open_system('systemname')

　　open_system('blockname')

　　open_system('blockname', 'force')

说明：

open_system('systemname')：打开指定的系统或者子系统窗口.

open_system('blockname')：打开与指定的模块相关联的对话框，其中'blockname'是完整的模块路径名. 如果定义了模块的 OpenFcn 回调函数，计算该程序.

open_system('blockname', 'force')：查看指定系统的模板内的情况，其中'blockname'是完整路径名或者模板系统. 该命令等价于使用 Look Under Mask 菜单项.

例 2.14 下面的命令在缺省的显示位置打开 controller 系统：

```
open_system('controller')
```

下面的命令打开 controller 系统内的 Gain 模块的对话框：

```
open_system('controller/Gain')
```

参见：close_system，new_system，save_system 命令

2.18.16 命令 replace_block

功能：在 Simulink 模型中替换模块.

语法：replace_block('systemname', 'blk1', 'blk2', 'noprompt')

replace_block('systemname', 'Parameter', 'value', 'blockname', ...)

说明：

replace_block('systemname', 'blk1', 'blk2', 'noprompt')；用 'blk2'替换 'systemname'系统中所有类型为 'blk1'的模块或者模板. 如果 'blk2'是 Simulink 的内建模块，只需要指明模块的名字. 如果 'blockname'在另外的系统中，需要指明完整的模块路径名. 如果没有带'noprompt'参数，Simulink 在替换之前显示一个对话框要求选择匹配的模块. 指定 'noprompte'参数就不会出现该对话框. 如果指定返回变量，被替换的模块的路径保存在那个变量中.

replace_block('systemname', 'Parameter', 'value', 'blockname', ...)；用 'blockname'模块替换 'systemname'系统中所有指定的参数具有指定值的模块. 可以指定任何数目的参数/值对.

值得注意的是，由于该命令所做的修改很难撤消，因此在替换前最好先保存系统.

例 2.15 下面的命令用 Integrator 模块替换 f14 系统中的所有 Gain 模块并且将被替换模块的路径保存在 RepNames 变量中. Simulink 在替换之前列出匹配的模块：

RepNames=replace_block('f14', 'Gain', 'Integrator')

下面的命令用 Integrator 模块替换 clutch 系统的 Unlocked 子系统中所有 Gain 参数为'bv'的模块. Simulink 在替换之前显示一个对话框列出匹配的模块：

replace_block('clutch/Unlocked', 'Gain', 'bv', 'Integrator')

下面的命令用 Integrator 模块替换 f14 系统中的所有 Gain 模块，不显示对话框：

replace_block('f14', 'Gain', 'Integrator', 'noprompt')

参见：find_system，set_param 命令

2.18.17 命令 save_system

功能：保存 Simulink 系统.

语法：save_system

save_system('systemname')

save_system('systemname', 'newname')

说明：

save_system；用当前的名字保存当前的顶层系统到文件中.

save_system('systemname')；用当前的名字保存指定的顶层系统到文件. 该系统必须已经打开.

save_system('systemname', 'newname')；用指定的新的名字保存指定的顶层系统到文件. 该系统必须已经打开.

例 2.16 下面的命令保存当前系统：

save_system

下面的命令保存 vdp 系统：

save_system('vdp')

下面的命令用'myvdp'的名字保存 vdp 系统到文件：

save_system('vdp', 'myvdp')

参见：close_system, new_system, open_system 命令

2.18.18 命令 set_param

功能：设置 Simulink 系统和模块参数.

语法：set_param('objectname', 'parameter1', value1, 'parameter2', value2, ...)

说明：

set_param('objectname', 'parameter1', value1, 'parameter2', value2, ...)：用指定的值设置指定的参数，其中 'objectname' 是系统或模块的路径. 忽略参数名的大小写. 参数值字符串区分大小写. 任何与对话框的条目相应的参数具有字符串形式的值.

可以在仿真期间在工作空间中改变模块参数的值并且用这些改动更新模块图. 要做到这一点，在命令窗口中做这些改动，接着使模型窗口成为活动窗口，然后从Edit 菜单中选择 Update Diagram 菜单项.

大多数模块参数值必须以字符串的形式指定，除了所有模块共有的 Position 和 UserData 参数以外.

例 2.17 下面的命令设置 vdp 系统的 Solver 和 StopTime 参数：

set_param('vdp', 'Solver', 'ode15s', 'StopTime', '3000')

下面的命令设置 vdp 系统中 Mu 模块的 Gain 为 1000：

set_param('vdp/Mu', 'Gain', 1000)

下面的命令设置 vdp 系统中 Fcn 模块的位置：

set_param('vdp/Fcn', 'Position', [50 100 110 120])

下面的命令设置 mymodel 系统中 Zero-Pole 模块的 Zeros 和 Poles 参数：

set_param('mymodel/Zero-Pole', 'Zeros', '[2 4]', 'Poles', '[1 2 3]')

下面的命令设置模板子系统中的一个模块的 Gain 参数，变量 k 与 Gain 参数相关联：

set_param('mymodel/Subsystem', 'k', '10')

下面的命令设置 mymodel 系统中名为 Compute 的模块的 OpenFcn 回调参数. 当用户双击 Compute 模块时执行 'my_open_fcn'函数.

set_param('mymodel/Compute', 'OpenFcn', 'my_open_fcn')

参见：get_param, find_system 命令

2.18.19 命令 Simulink

功能：打开 Simulink 模块库.

语法：simulink

说明：

Simulink 命令打开 Simulink 模块库窗口，创建且显示一个新的空模型窗口，下列两种情况除外：

1）如果已经打开了一个模型窗口，Simulink 不创建新的模型窗口.

2）如果已经打开了 Simulink 模块库窗口，发布该条命令，使 Simulink 窗口成为活动窗口.

关闭所有 Simulink 窗口，即可结束仿真任务. 在 Windows 操作系统平台上，选择File 菜单下的 Exit MATLAB 命令；UNIX 系统平台上，选择File 菜单下的Quit MATLAB 命令.

第三章　使用模板定制模块及条件执行子系统

制作模板是 Simulink 的一个非常强大的功能，通过它可以为一个子系统定制对话框和图标. 制作模板，具有以下功能：

1) 通过仅用一个对话框来代替子系统中各个模块的对话框，以简化模型的使用. 这样模型的用户就不再需要打开每一个模块对话框以输入参数值，这些参数值可以在模板对话框里输入就行了，它们会被传给模板子系统中的各模块.

2) 用自己的模块描述、参数域标注和帮助文本定义对话框，将会提供一个更能说明问题、更有用的用户界面.

3) 定义计算变量的命令，这些变量的值取决于模块的参数.

4) 创建描绘子系统的功能的模块图标.

5) 通过将子系统的内容隐藏在定制的用户界面的后面，防止对子系统无意的改动.

6) 创建动态对话框.

3.1　示例模板子系统

下面通过一个例子来说明如何制作子系统的模板. 这个例子给一条直线的方程建模，直线方程为 $y=mx+b$.

如果在子系统模块(图 3.1(a))上双击鼠标时，将会打开子系统模块，在另一个单独的窗口中显示出子系统中的各个模块(图 3.1(b)). $y=mx+b$ 子系统包含一个 Gain 模块，取名为斜率，它的 Gain 参数为 m，还包含一个 Constant 模块，取名为截距，它的 Constant 参数用 b 来表示. 这些参数用来表示一条直线的斜率和截距.

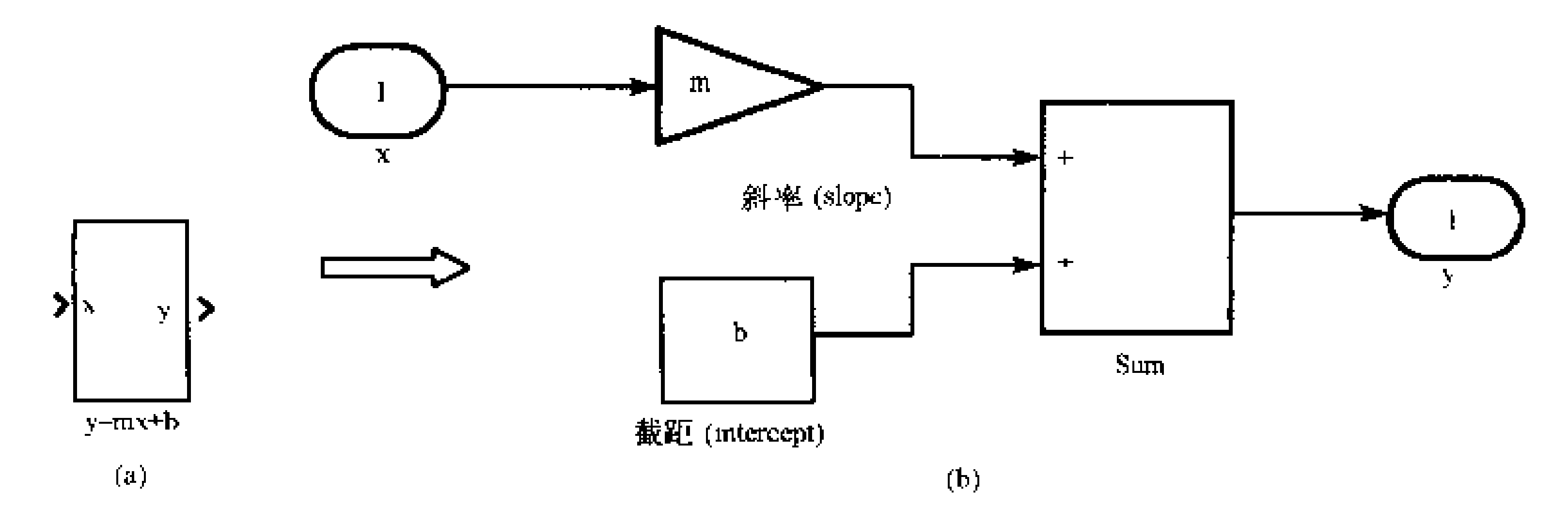


图 3.1　模型 $y=mx+b$ 模块图

在该示例中，为子系统创建一个定制的对话框，对话框中包括斜率和截距的有关提示. 在创建了模块以后，双击子系统模板将会打开模块参数对话框. 模块参数对话框如图 3.2 所示.

子系统的用户只用在模板对话框中输入斜率和截距的值就行了. 子系统下的所有模

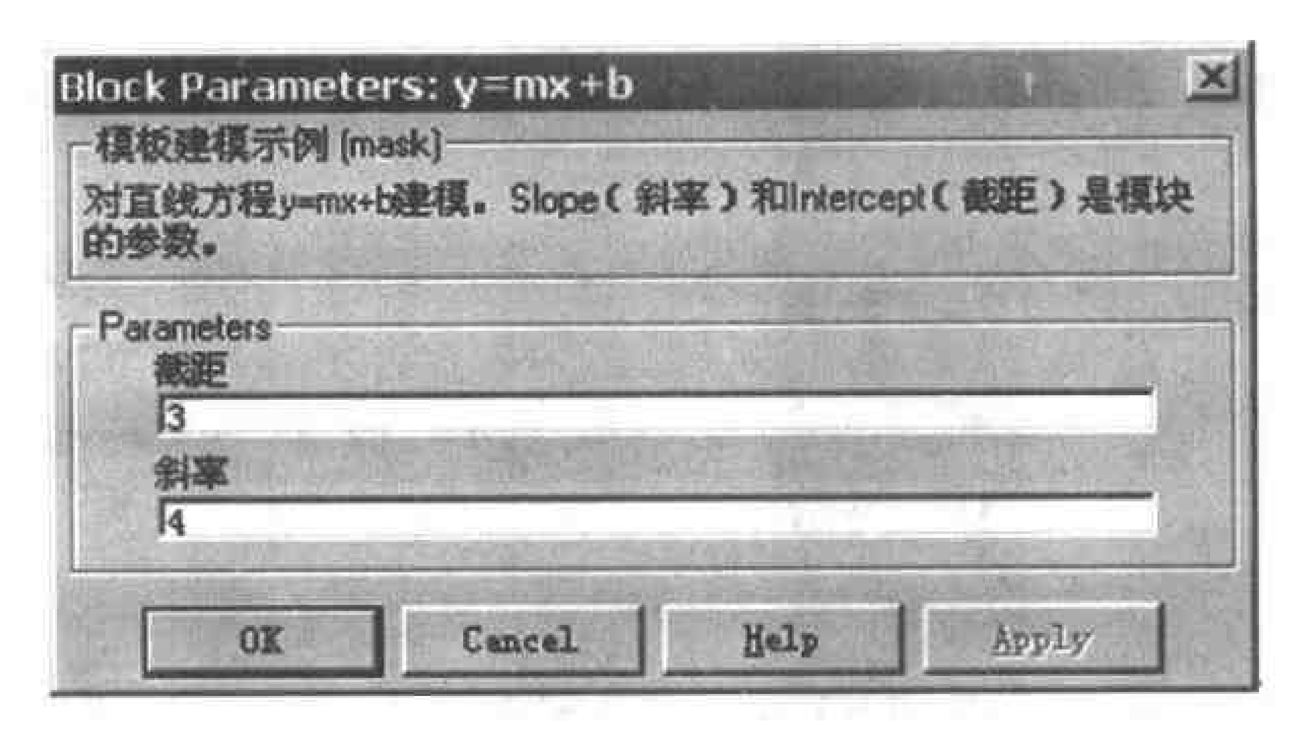


图 3.2 新建模块的参数对话框

块都可访问这些值，模板将模板参数映到它的下层模块的普通参数上. 子系统的复杂性被一个新的界面封装起来，并且它看起来好像是一个 Simulink 的内建模块.

为该子系统创建模板包含如下的工作：

1）指定模板对话框中各参数的提示，在该示例中，模板对话框有对斜率和截距的提示.

2）指定保存各参数值的变量名.

3）为模块输入有关资料文档，包括模块的说明和帮助文本.

4）指定创建模块图标的绘图命令.

5）指定提供绘图时所需变量的命令.

3.1.1 创建模板对话框提示

要创建该子系统的模板，选取子系统模块后，从Edit 菜单下选择Mask Subsystem 或 Edit Mask 菜单项.

前面给出的模板对话框主要是通过模板编辑器(Mask Editor)的 Initialization 页面实现的，如图 3.3 所示.

通过模板编辑器可以指定模板参数的如下属性：

1）提示——描述参数的文本标注.

2）控件类型——决定如何输入或选择参数值的用户界面的控件类型.

3）变量——用来保存参数值的变量的名称.

一般来讲，有了模板参数的提示是比较方便的. 在该示例中，与斜率有关的参数的提示是“斜率”，与截距有关的参数的提示是“截距”.

用来输入斜率和截距参数值的都被定义为编辑框控件. 这意味着用户可以在模板对话框的编辑框中输入它们的值. 这些参数值保存在模板工作空间的变量中. 模板模块只能访问模板工作空间中的变量. 在该示例中，输入的斜率值被赋给了变量 m. 模板子系统中的 Slope 模块从模板工作空间中获取斜率参数的值.

在定义好斜率和截距的模板参数以后，按 OK 按钮，然后双击该子系统模块以打开新

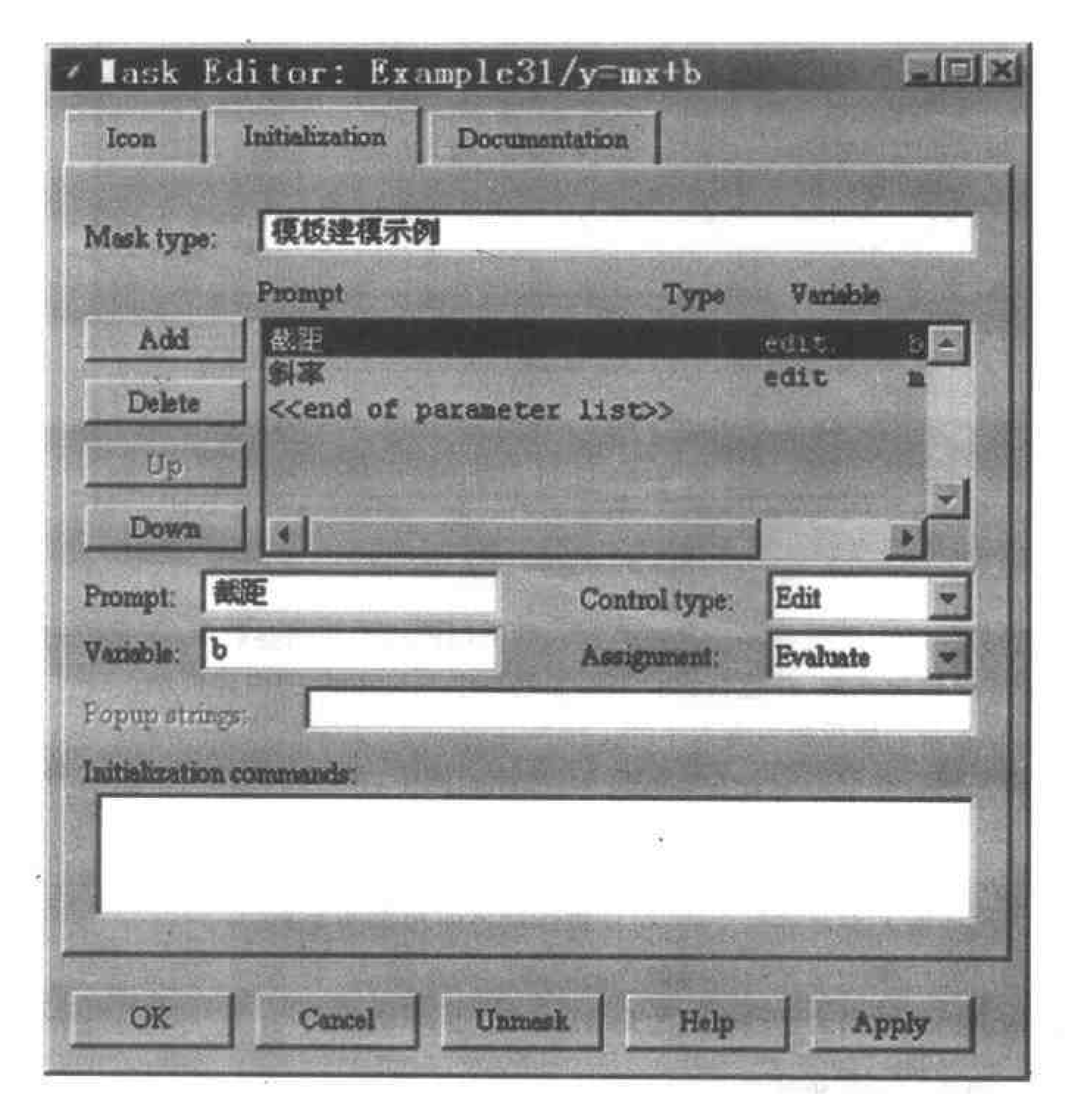


图 3.3 模板编辑器的 Initialization 页面

构建的对话框，如图 3.2 所示，其中截距参数值为 3，斜率参数值为 4.

3.1.2 创建模块的描述和帮助文本

在模板编辑器窗口(Mask Editor)的 Documentation 页面中，可以定义模板类型、模块描述和帮助文本. 对于该示例的模板模块，该页面如图 3.4 所示.

3.1.3 创建模块图标

现在，已经为 y=mx+b 子系统创建好了定制的对话框，然而子系统模块显示的仍然是普通的 Simulink 子系统图标. 该模板模块的比较形象的图标应该是表示直线斜率的图形. 斜率为 4 时，图标如图 3.5 所示.

模块的图标可以在模板编辑器窗口的 Icon 页中定义. 要得到如图 3.5 所示的图标，该模块的 Icon 页如图 3.6 所示.

绘图命令绘一条从(0,0)到(0,m)的直线. 如果斜率是负的，直线向上移动，以保证它在模块的可见绘图区域内.

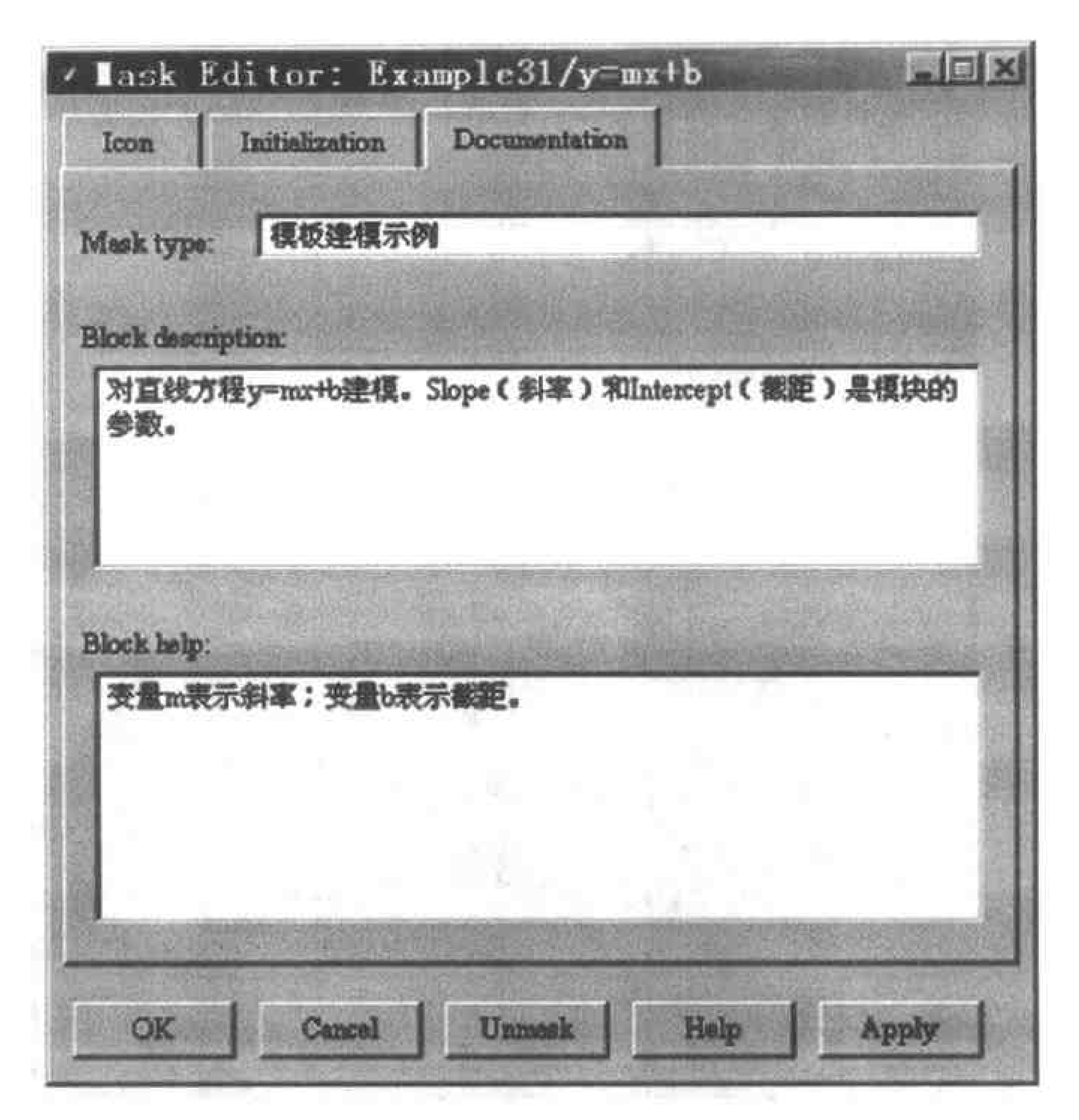


图 3.4　模板编辑器的 Documentation 页面

绘图命令能够访问模板工作空间中的所有变量，因此当输入不同的斜率值时，图标自动地重新绘制直线.

当选择 Normalized 作为 Drawing coordinates 参数（在图标属性列表的底部）的值时，指定绘制图标的画面的左下角为(0,0)，右上角为(1,1).

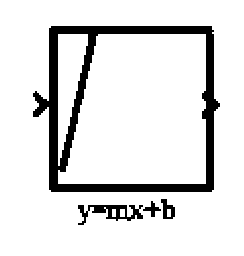


图 3.5　创建后图标显示

3.1.4　创建模板步骤

创建一个模板通常包括如下步骤：

1）定义对话框的提示及其特征.

2）定义模板模块的说明和帮助文本.

3）定义创建模板模块图标的命令.

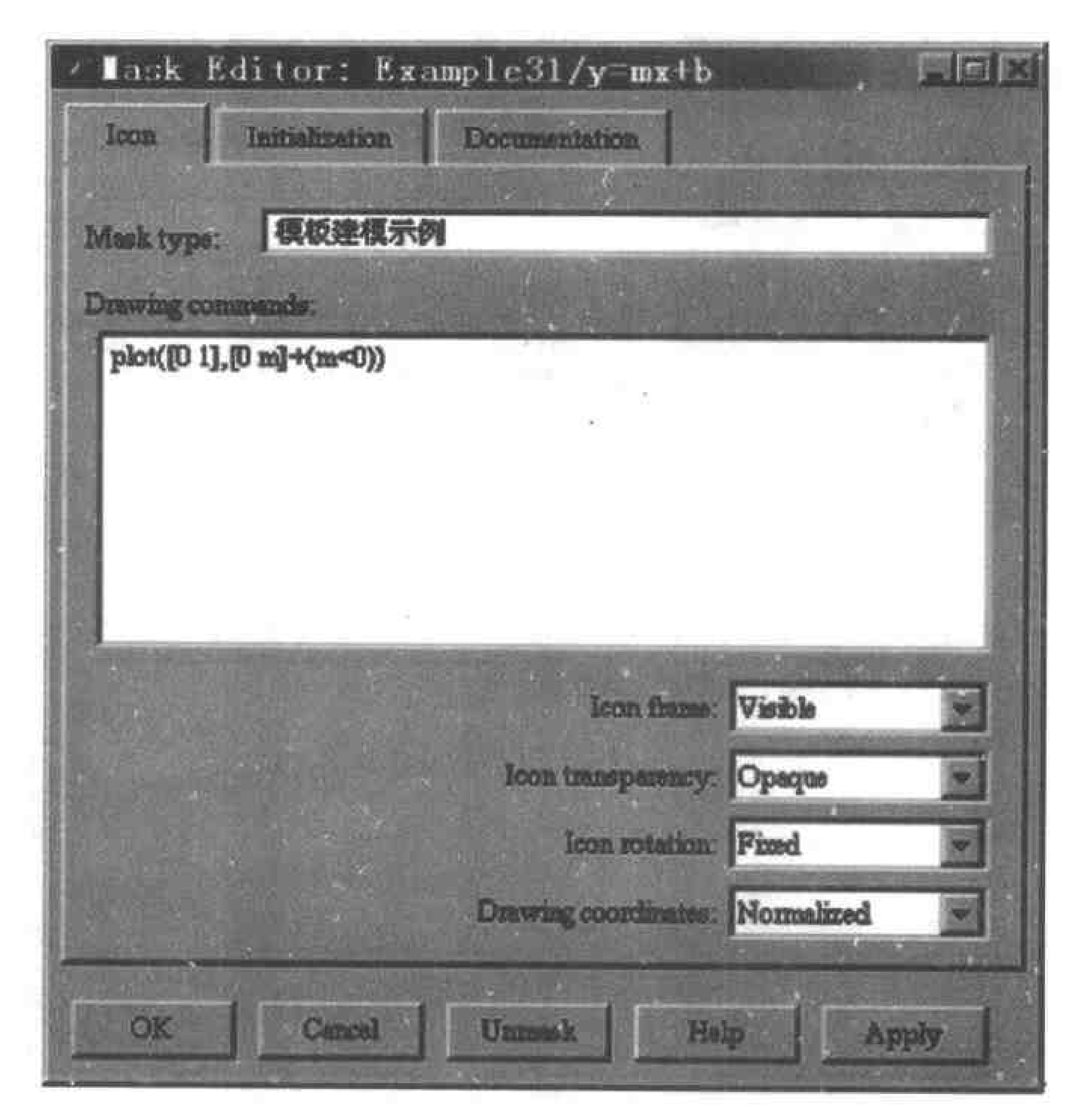


图 3.6　在模板编辑窗口创建图标

3.2　模板编辑器

要制作一个子系统的模板(只能对 Subsystem 模块制作模板),选取要制作模板的 Subsystem 模块,然后选择Edit 菜单下的Mask Subsystem 菜单项,将会显示模板编辑器.模板编辑器由三个页面组成:

1) Initialization 页面,在该页面中可以定义和描绘模板对话框的参数提示,与参数有关的变量的名字并且给定初始化命令.

2) Icon 页面,在该页面中可以定义模块的图标.

3) Documentation 页面,在该页面中可以定义模板的类型,并给出模块的描述和模块的帮助.

另外,在模板编辑器的底部显示有五个按钮:

1) OK 按钮,点击该按钮使用所有页面中新设定的模板参数,并关闭模板编辑器.

2) Cancel 按钮,点击该按钮关闭模板编辑器,不使用各页面中新设定的模板参数.

3) Unmask 按钮,点击该按钮解除模板并关闭模板编辑器.模板的有关信息还保存着以便可以重新启用模板.要重新启用模板,选取模块后选择 Create Mask.模板编辑器打开并显示当初的设置.如果模型被关闭的话,非活动模板信息将会丢失而不能恢复.

4) Help 按钮,点击该按钮将会显示有关帮助内容.

5) Apply 按钮,点击该按钮将使用模板编辑器中所有页面内提供的信息创建或改变模板.模板编辑器保持打开状态.

要查看一个模板形式的系统的内容,但不能打散该模板,选取要查看的 Subsystem 模块,然后选择Edit 菜单下的 Look Under Mask 子菜单项.这一命令打开子系统,但模块的模板不受影响.

3.2.1 Initialization 页

模板对话框界面使得模板系统的用户可以为模板系统中的模块输入参数值.通过在 Initialization 页中定义参数值的提示创建模板对话框界面.示例模板系统 y=mx+b 的 Initialization 页面如图 3.3 所示.

3.2.1.1 提示及相关变量

提示(prompt)提供的有关信息,可以帮助用户为模块输入或选择参数值.显示在模板对话框中提示的顺序(如图 3.2)与显示在 Prompt 列表(如图 3.3)中的相同.

在定义一个提示时,必须指定一个变量以保存该参数值,必须选择与该提示相关的控件类型,并指明参数值如何保存于变量当中.

如果 Assignment 是 Evaluate,用户输入的值在赋给变量之前会先由 MATLAB 计算出来.

如果是 Literal,用户输入的值不会先被计算,而是作为一个字符串赋值给变量.

如果用户在编辑框中输入字符串 gain,且 Assignment 类型是 Evaluate,MATLAB 将会计算 gain 的值并将结果赋给变量;而如果类型是 Literal,MATLAB 不会求它的值,所以变量包含字符串"gain".

如果既需要字符串,又需要求它的值,选择 Literal,然后在初始化命令中用 MATLAB 的 eval 命令.如果 LitVal 是字符"gain",可以用下面的命令求得它的值:

value=eval(LitVal)

通常大多数参数使用的 Assignment 类型为 Evaluate.

1) 创建第一个提示.要创建提示列表中的第一个提示,在 Prompt 域输入提示,在 Variable 域输入要保存参数值的变量名,并选择控件类型和赋值方式.

2) 插入提示.要在提示列表中插入提示,首先选取将要在其前面插入新的提示的"提示",然后点击提示列表左边的 Add 按钮;在 Prompt 域中输入要插入的提示的文本,在 Variable 域中输入用来保存该参数值的变量.

3) 编辑提示.要编辑一个已存在的提示,首先在列表中选取要编辑的"提示".提示、变量名、控件类型和赋值方式都会显示在列表的下方;编辑合适的值,如果在其它的区域点击鼠标或敲 Enter 键,Simulink 会更新提示.

4) 删除提示.要从列表中删除提示,首先选取要删除的"提示";点击提示列表左边的

Delete 按钮.

5) 移动提示. 要在列表中移动提示，首先选取要移动的提示；要将提示在提示列表中往上移动，点击提示列表左边的 Up 按钮；要往下移动，点击 Down 按钮.

3.2.1.2 控件类型

Simulink 可以让用户选择参数值输入或选择的方法. 用户可以创建三种类型的控件：编辑框控件、检查框控件和弹出菜单控件. 图 3.7 所示的是一个模板的对话框，该对话框中用到了前面所说的三种类型的控件.



图 3.7 在一个模板对话框中有三种控件

(1) 定义编辑框控件

编辑框使得用户可以从中输入参数值. 图 3.8 显示了如何在模板编辑器中定义编辑框控件.

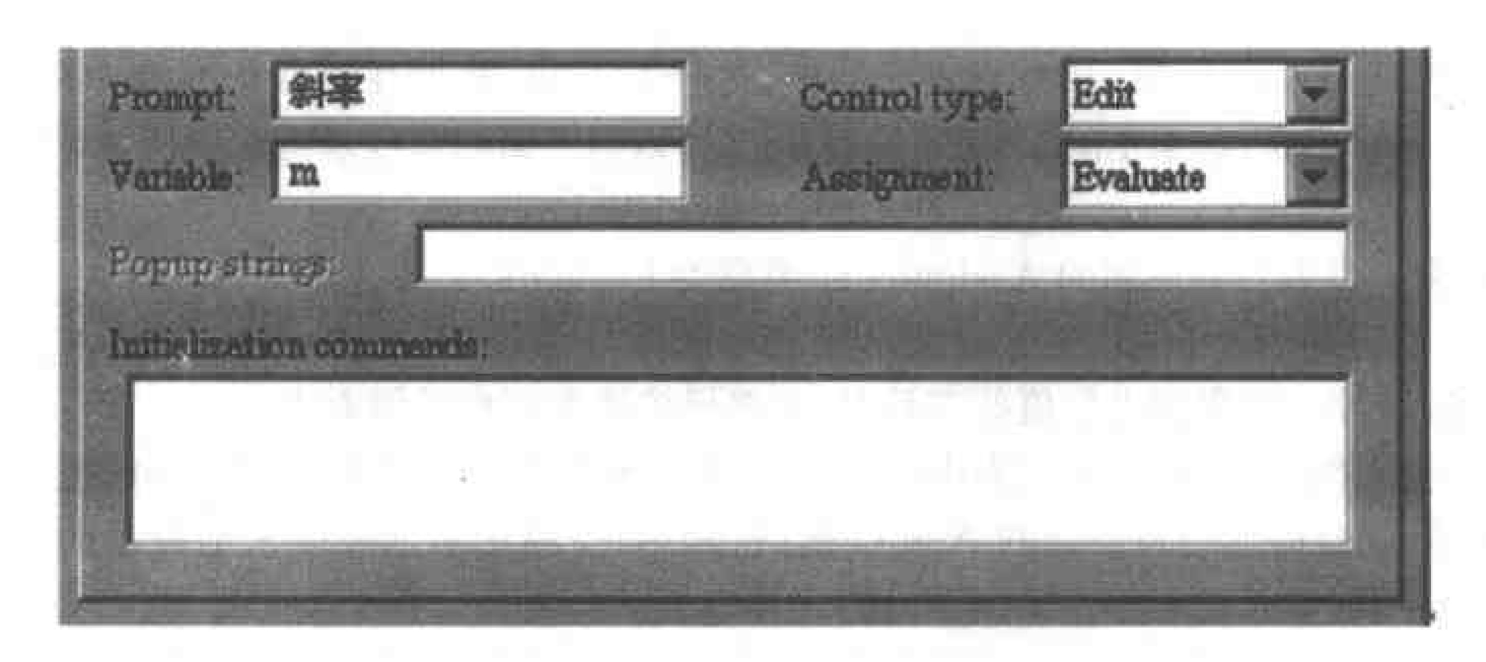


图 3.8 定义编辑框控件

与参数"斜率"相关的变量 m 的值由该提示的 Assignment 的类型决定：当 Assignment 为 Evaluate 时，变量值为输入域表达式计算结果；当 Assignment 为 Literal 时，变量

值为输入域实际输入的字符串.

(2) 定义检查框控件

检查框使得用户可以在选与不选该检查框两者之间选择其一.图3.9显示了如何在模板编辑器中定义检查框控件.

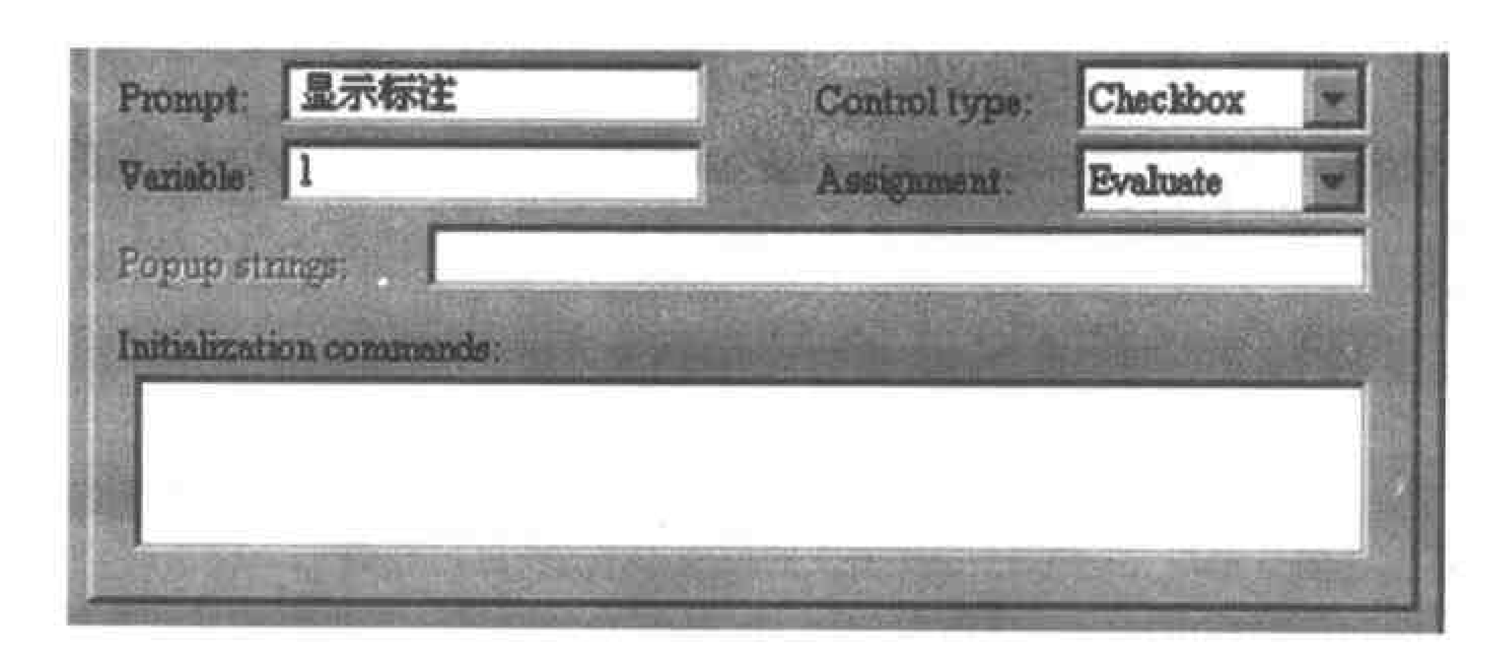


图3.9　定义检查框控件

与参数"显示标注"相关的变量l的值,取决于检查框是否被选中,以及Assignment的类型.表3.1列出其相互关系.

表3.1　检查框变量取值

检查框状态	Evaluated	Literal
选中	1	'on'
没有选中	0	'off'

(3) 定义弹出式菜单控件

弹出式菜单使得用户可以为参数在多种可能的值中间选取一个值.可以从Popup strings域中给出选项列,各选项之间用竖线(|)分开.图3.10显示了如何在模板编辑器中定义弹出式菜单控件.

与参数"颜色"相关的变量c的值,取决于从弹出式菜单中选择了哪一项,以及

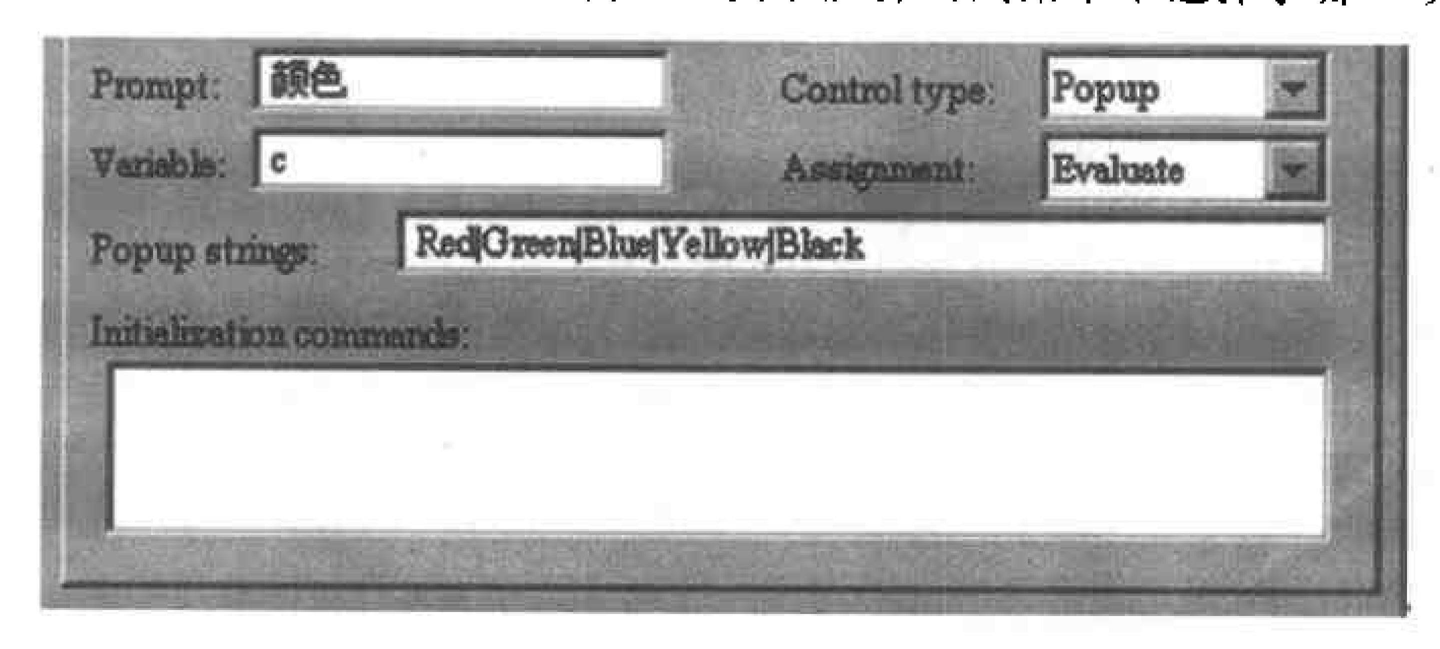


图3.10　定义弹出式菜单控件

Assignment的类型. 如果 Assignment 为 Evaluate，则从弹出菜单的选项列中选择的那一项的索引，第一项的索引为 1，选择了第三项，则参数的值为 3；如果 Assignment 为 Literal，c 值为被选择的字符串，如果选择了第三项，则参数值为"Blue".

3.2.1.3 模板模块参数的缺省值

要在一个模板库模块中改变缺省参数值，按如下步骤进行：

1) 取消库的锁定状态.

2) 打开模块以访问其对话框，填好各个期望的缺省值后关闭对话框.

3) 保存库.

当模块被拷贝到一个模型中并打开时，在模块的对话框中将显示缺省值.

3.2.1.4 可调参数

可调参数就是用户在运算时可以改变的模板参数. 当用户创建一个模块时，所有其参数都是可调的. 随后，可以通过使用 MaskTunableValues 参数，停止或激活任何模板参数的可调性，该参数是一个字符串单元数组，每个单元对应一个模板模块参数. 第一个单元对应于第一个参数，第二个单元对应于第二参数，依次类推. 如果一个参数是可调的，则相应单元的值为"on"，否则为"off". 激活或停止一个参数的可调性，首先用 get_param 获取单元数组，然后，设置相应的单元为"on"或"off"，并用 get_param 重置 MaskTunableValues 参数. 如下列命令停止当前所选模板模块的第一个参数的可调性.

```
ca = get_param(gcb, 'MaskTunableValues');
ca(1) = 'off'
set_param(gcb, 'MaskTunableValues', ca)
```

改变参数可调性后，保存模块，就使其永远改变了.

3.2.1.5 初始化命令

初始化命令定义驻留在模板工作空间中的变量. 这些变量可以用于所有为模板定义的初始化命令，模板子系统中的所有模块和绘制模块图标的命令.

在下列情况出现时，Simulink 执行初始化命令：

1) 模型被装入时.

2) 仿真开始或模块图被更新时.

3) 模板模块被旋转时.

4) 模块的图标需要重画，且绘图命令 plot 需要用到初始化命令中定义的变量.

初始化命令应该是合法的 MATLAB 表达式，可以由 MATLAB 的函数、运算符和在模板工作空间中定义的变量构成. 初始化命令不能访问底部工作空间的变量. 初始化命令用分号结束，可以防止在命令窗口中显示结果.

(1) 模板工作空间(mask workspace)

当下列任一情况出现时，Simulink 创建一个被称为模板工作空间的局部工作空间：

1) 模板包含有初始化命令.

2) 模板定义了提示，并具有与这些提示相关联变量.

模板模块不能访问底层工作空间或其它模板工作空间.

模板工作空间的内容包括与模板的参数有关的变量和初始化命令中定义的变量.在模板工作空间中的这些变量可以被模板模块访问.如果模块是一个子系统,那么它们还可以被子系统中的所有模块访问.

模板工作空间与 M 文件函数用到的局部工作空间相似.可以将低层模块的对话框中输入的表达式和在模板编辑器中输入的初始化命令看作一个 M 文件中的命令行.如果这么类比,该"函数"的局部工作空间是模板工作空间.

在前面举的例子 $y=mx+b$ 中,模板编辑器通过将变量与模板参数相联系而在模板工作空间中明确地创建了两个变量 m 和 b,然而模板工作空间中的变量并不明确地将这些变量赋给模板低层的各个模块,但是模板内的各个模块可以访问工作空间中的变量.

图 3.11 显示了在模板对话框中给模板工作空间中的变量输入的值和下层模块访问这些变量的对应关系.

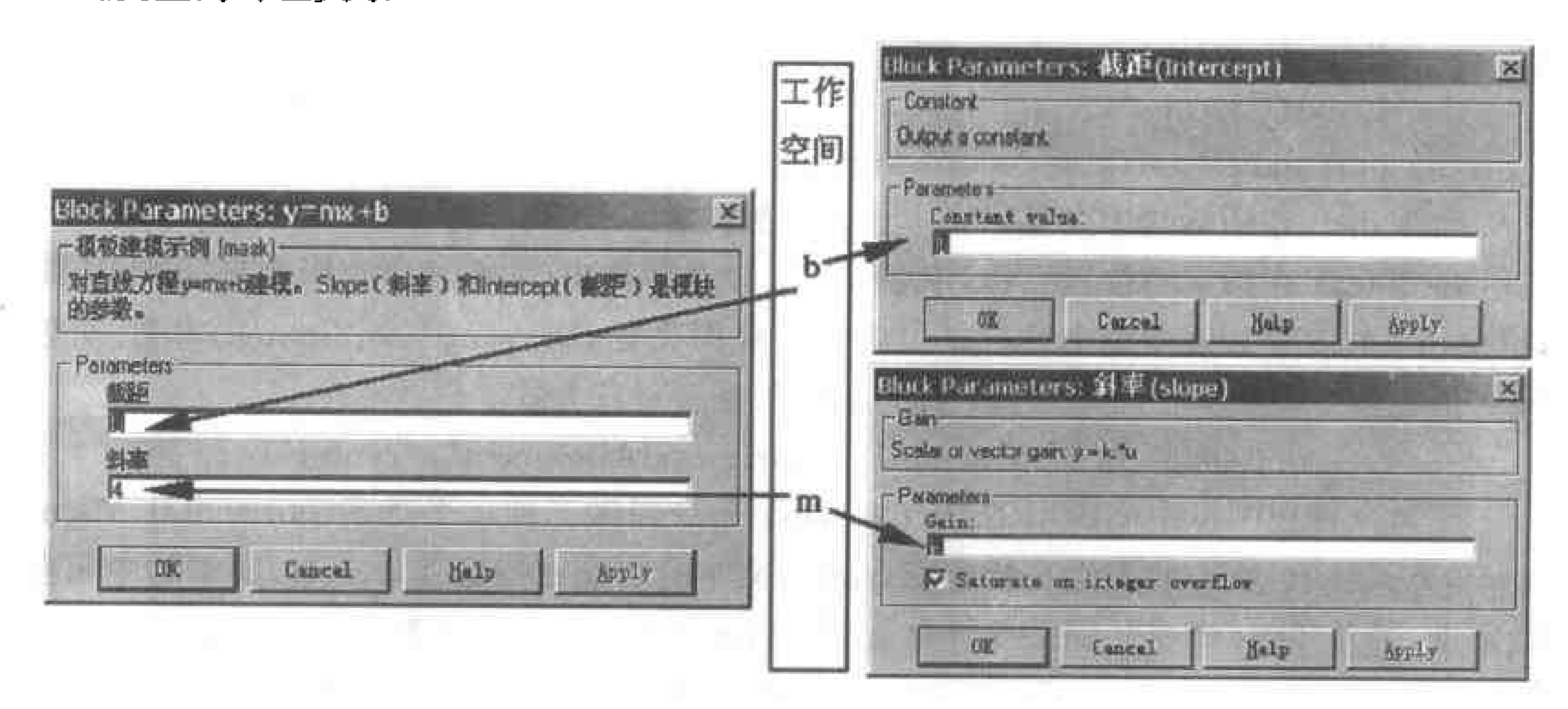


图 3.11 模板工作空间与下层模块变量的关系

(2) 调试初始化命令

可以通过如下方法调试初始化命令:

1) 给定初始化命令时,不用分号(";")结束,以便运行时它的结果显示在命令窗口中.

2) 在初始化命令中设置一条键盘操作命令,以暂停执行,并将控制交给键盘.

3) 在 MATLAB 的命令窗口中输入下面两条命令中的任一条.

```
dbstop if error
dbstop if warning
```

如果在初始化命令中出现错误,执行将停止,这时可以检查模板工作空间.

3.2.2 Icon 页

在模板编辑器的 Icon 页中,可以定制模板模块的图标.可以在 Drawing commands 域中设定绘制图标的命令.可以在图标中显示描述文本、状态方程、图像和图形.图 3.6 显示了模板编辑器 Icon 页.

绘图命令可以访问模板工作空间中的所有变量.

绘图命令可以显示:文本,一幅或多幅图,或者是传递函数. 如果输入了多条命令,命令的结果是按命令出现的次序出现在图标中的.

3.2.2.1 在模块图标中显示文本

要在图标中显示文本,可以在 Drawing commands 域中输入一条或多条绘图命令. 这些命令有:

```
disp('text')
disp(variablename)
text(x,y,'text')
text(x, y, stringvariablename)
text(x, y, text, 'horizontalAlignment, 'halign, 'verticalAlignment, 'valign)
fprintf('text')
fprintf('format', variablename)
port_label(port_type,port_number,label)
```

命令 disp 在图标的中央显示文本或变量的内容.

命令 text 在指定的位置点(x, y)处,放上一个字符串(文本或字符串变量 stringvariablename 的内容). 该命令中坐标单位依赖于绘图坐标(Drawing coordinates)参数. 可以随意指定命令中文本相对于点(x, y)的水平与垂直对齐方式.

```
text(0.5, 0.5, 'Example', 'horizontalAlignment', 'center')
```

该命令将在图标中间显示 Example. Text 命令提供三种水平对齐(horizontal Alignment)方式:left,文本的左端在指定点;right,文本的右端在指定点;center,文本的中间在指定点. 另外 Text 命令还提供五种垂直对齐(verticalAlignment)方式,见表 3.2.

表 3.2 Text 命令提供的垂直对齐方式

verticalAlignment 选项	对齐方式
base	文本基线在指定点
bottom	文本底线在指定点
middle	文本中线在指定点
cap	文本的大写字母线在指定点
top	文本的顶部在指定点

命令 fprintf 在图标的中央显示格式化的文本,并能将文本与变量 variablename 的内容显示在一起.

要注意的是,如果这些命令与 MATLAB 相应的函数的名字相同,它们只提供上面介绍的功能. 要显示多行文本,用"\n"表示换行. 在图标中显示的文本如图 3.12,命令 disp('示例图标文本'),见左图;disp('示例\n 图标文本'),见右图.

port_label 命令可以让用户指定图标显示的端口的标注. 命令为

```
port_label(port_type, port_number, label)
```

其中,port_type 可为 'input' 或 'output';port_number 可为一整数;而 label 是一个指定

端口标注的字符串. 如:

port_label('input',1, 'a')

是定义输入端口 1 的标注为 a.

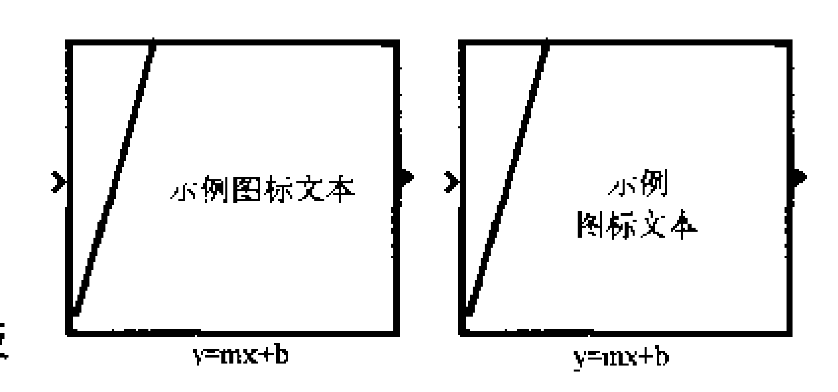


图 3.12　图标中显示文本

3.2.2.2　在模块图标中显示图形

可以输入一条或多条绘图命令,在模板模块的图标中显示绘制的图形. 可以使用绘图命令的如下形式:

plot(y)

plot(x1, y1, x2, y2)

如果 y 是一向量,plot(y)按顺序绘制 y 中的每一个元素,如果 y 是一矩阵,它将矩阵中的每一列作为一个向量来分别绘出.

plot(x1,y1,x2,y2,...)以向量 y1 对 x1,y2 对 x2,...,分别绘图. 向量对的长度必须相等,并且向量的个数必须是偶数.

例 3.1　下面的命令生成 Sources 库中 Ramp 模块的图标,如图 3.13 所示.

plot([0　1　5], [0　0　4])

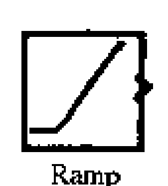


图 3.13　Ramp 模块的图标

绘图命令中可以包含 NaN 和 inf 值. 当遇到 NaN 或 inf 时,Simulink 停止绘图,然后在下一个非 NaN 或 inf 数处重新开始绘图.

在图标中绘的图的外观依赖于 Drawing coordinates 参数的值.

在遇到如下情况时,Simulink 显示三个问号(???)并显示警告信息:

1) 当绘图命令中使用的参数值还没有定义时,例如,模板已创建好,但还没有在模板对话框中输入值.

2) 当模板模块的参数或绘图命令输入不正确时.

3.2.2.3　在模板中显示图像

使用 image 和 patch 命令,可以在模板模块图标中显示位图像或画图像. image (I),可显示图像 I,I 是一个 M×N×3 的 RGB 值数组. 用户可以用 imread 和 ind2rgb 命令读入位图图像,并转换成必要的矩阵格式.

image(a, [x, y, w, h]),从模板左下角开始,在指定的位置产生图像.

image (a, [x, y, w, h], rotation),允许指定是否对图像进行旋转. rotation 选'on'时,旋转;选'off'时,不旋转;缺省值为'off'.

patch(x, y),产生由坐标向量(x, y)指定形状的实体块,实体块的颜色为当前的前景颜色.

patch(x, y, [r g b]),产生由坐标向量(x, y)指定形状,颜色向量[r g b]指定颜色的实体块. 其中 r 为红色成分;g 为绿色成分;b 为蓝色成分. 如:

patch([0 .5 1], [0 1 0], [1 0 0])

在模板图标中画一个红色三角形.

3.2.2.4　在模块图标中显示传递函数

要在模块图标中显示传递函数，在 Drawing commands 域中输入如下命令：

> matlab7中拖放了Transfer Fcn模块后点右键选择“Transfer Fcn Parameters,在弹出的对话框中设置即可

dpoly(numerator, denominator)

dpoly(numerator, denominator, 'character')

其中 numerator 和 denominator 是表示传递函数的分子和分母系数的向量，它们通常用初始化命令定义. 传递函数的方程用指定的字符 character 表达，缺省为 s，当画图标时，初始化命令被执行并将方程的结果显示在图标中.

1) 要显示一个以 s 的降幂表示的连续传递函数，输入如下命令：

dpoly(numerator, denominator)

例 3.2　当 numerator＝[1　0　1]，且 denominator＝[1　2　1]时，图标如图 3.14 所示.

2) 要显示一个以 z 的降幂表示的离散传递函数，输入如下命令：

dpoly(numerator, denominator, 'z')



图 3.14　在图标中显示变量 s 的传递函数　　　图 3.15　在图标中显示离散传递函数

例 3.3　当 numerator＝[1　0　1]，denominator＝[1　2　1]时，图标如图 3.15 所示.

3) 要显示一个以 1/z 的升幂表示的传递函数，输入

dpoly(numerator, denominator, 'z－')

例 3.4　当 numerator＝[1　0　1]，denominator＝[1　2　1]时，图标如图 3.16 所示.

4) 要显示一个零-极点增益传递函数，输入如下命令：

droots(z, p, k)



图 3.16　在图标中显示 1/z 表示的传递函数　　　图 3.17　显示零-极点增益传递函数

例 3.5　当上面的命令中 z＝[－2]，p＝[－1　1]，k＝2 时，创建的图标如图 3.17 所示.

可以在 droots 命令中指定第四个参数('z'或'z－')，以确定方程是用 z 还是用 1/z 表达.

在创建图标时，如果参数还没有定义或没有赋值，Simulink 将在图标中显示三个问号(???). 当在模板对话框中输入参数值时，Simulink 计算传递函数并且将方程的结果显

示在图标中.

3.2.2.5 控制图标属性

可以通过在模板编辑器的图标页 Drawing commands 域下面，选择某些选项来控制模板模块的图标属性. 这些图标属性包括：图标边框、图标透明性、图标旋转、绘图坐标.

(1) 图标边框(icon frame)

图标的边框是包围模块的矩形框. 可以通过设定 Icon frame 参数为 Visible 或者 Invisible 来显示或隐藏边框. 缺省的是使图标边框可见. 例如，图 3.18 显示了一个与门(AND gate)模块图标边框的可见与不可见两种状态.



图 3.18 设置图标边框为可见与不可见　　图 3.19 设置图标的透明性

(2) 图标透明性(icon transparency)

图标可被设成不透明的(opaque)或透明的(transparent)，以隐藏或显示图标后面的区域. 缺省的是 Opaque，它会覆盖 Simulink 显示的一些信息，如端口标注. 图 3.19 显示一个 AND gate 模块的不透明和透明两种状态的图标.

(3) 图标旋转(icon rotation)

当模块被旋转或翻转时，可以选择是否旋转或翻转它的图标，或让它保持它的初始方向. 缺省的是不旋转图标. 图标的旋转与模块端口的旋转是一致的. 图 3.20 显示的是当旋转 AND gate 模块时，选择固定(fixed)和旋转(rotates)两种状态时的结果.

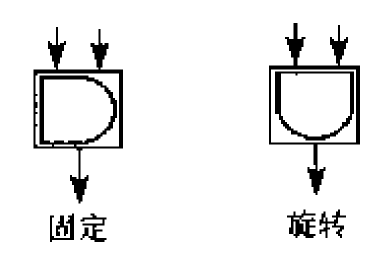


图 3.20 设置图标的旋转属性

(4) 绘图坐标(drawing coordinates)

这一参数控制绘图命令中用到的坐标系. 这一参数只适用于 plot 和 text 命令. 可以从下面三种选项中选择其一：Autoscale，Normalized 和 Pixel. 三种坐标如图 3.21 所示.

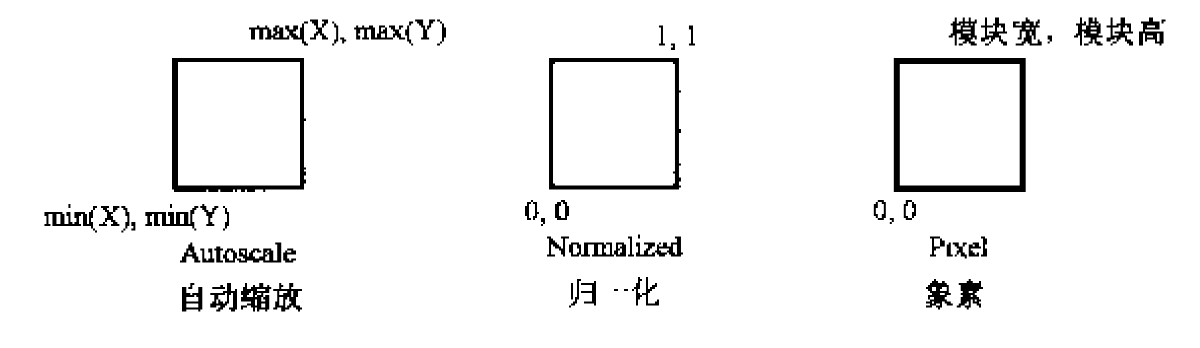


图 3.21 三种绘图坐标

1) 自动缩放. 在模块边框内自动缩放图标. 当模块被改变大小时，图标的大小也跟着改变.

例 3.6 图 3.22 所示的图标是用如下命令绘制的：

```
plot([0  2  4  6  8],[6  8  3  2  4]);
```

图 3.22 中，模块边框的左下角是(0,2)，右上角是(8,8). X 轴的范围是 8(从 0 到 8)，而 Y 轴的范围是 6(从 2 到 8).

2) 归一化. 在左下角为(0,0)，右上角为(1,1)的模块边框内绘制图标. X 和 Y 的值必须在 0 到 1 之间. 当模块的大小被改变时，模块图标的大小也跟着改变.

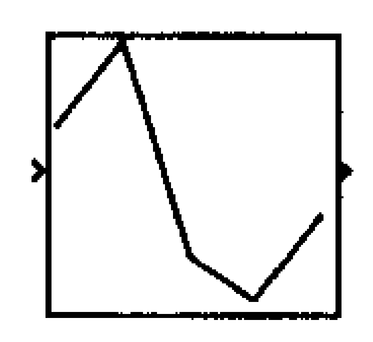


图 3.22 使用 autoscale 选项绘制图标

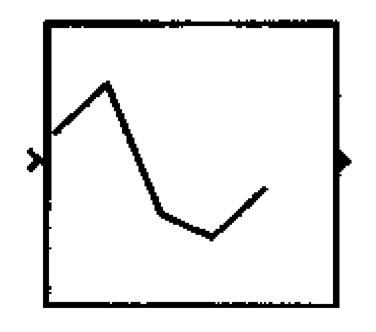


图 3.23 使用 normalized 选项绘制图标

例 3.7 图 3.23 所示的图标是用如下的命令绘制的：

```
plot([0  0.2  0.4  0.6  0.8],[0.6  0.8  0.3  0.2  0.4]);
```

3) 象素. 用以象素为单位的 X 和 Y 值绘制图标. 当模块的大小被改变时，图标的大小并不自动地改变. 要使图标的大小随着模块改变，用模块的大小定义绘图命令.

例 3.8 下面演示如何为前面讨论的 y=mx+b 示例模板子系统创建一个改进的图标. 这些初始化命令定义的数据，使得绘图命令能够生成一个精确的、与模块的形状无关的图标.

```
pos=get_param(gcb, 'Position');
width=pos(3)-pos(1);height=pos(4)-pos(2);
x=[0, width];
if(m>=0),y=[0,(m*width)];end
if(m<0),y=[height,(height+(m*width))];end
```

生成该图标的绘图命令是 plot(x,y).

3.2.3 Documentation 页

在模板编辑器的 Documentation 页中，可以定义或改变模板模块的类型、描述说明和帮助文本. 图 3.24 显示了 Documentation 页中的域是如何与 y=mx+b 示例模板模块的对话框相对应的.

3.2.3.1 模板类型域

模板类型是模块的分类，仅作为说明用. 它显示在模块的对话框和模块的所有 Mask Editor 页面中. 模板类型可以是任何名字. 当 Simulink 创建模块的对话框时，它在模板类型的后面加上"(mask)"，如图 3.2 所示，以将模板模块与内建模块区别开来.

3.2.3.2 模块描述域

模块描述是显示在模块对话框的模板类型下面的边框内的信息文本. 如果设计的系统要提供给别人使用，可以在这里描述模块的目的或功能.

如果文本中的行太长，Simulink 自动地对它进行折行，也可以用 Enter 或 Return 键

图 3.24　Documentation 页中的域和相关窗口的对应关系

强制换行.

3.2.3.3　模板的帮助文本域

当在模板模块的对话框中按 Help 按钮时，可以提供在其帮助窗口中显示的帮助文本. 如果创建的模型供别人使用，可在这里解释模块是如何工作的、有哪些参数以及如何输入参数.

可以包括用户写成的文件作为一个模板模块的帮助. 可以指定如下形式中的任何一种作为模板模块的帮助文本：

1) URL 地址；以 http:, www, file:, ftp:, mailto: 等字符开头的字符串.

2) web 命令；启动浏览器.

3) eval 命令；运算 MATLAB 字符串.

4) 在 Web 浏览器中，显示的静态文本.

Simulink 检查模板模块帮助文本的第一行. 如果它检测到了一个 URL 地址、web 命令或 eval 命令，它将根据它们的指引访问帮助文本；否则，模板模块帮助文本的所有内容显示在浏览器中.

例 3.9　下面这些例子都是能够接受的命令：

```
web([docroot '/My Blockset Doc/' get_param(gcb, 'MaskType') '.html'])
eval('! Word My_Spec.doc')
http://www.mathworks.com
file:///c:/mydir/helpdoc.html
```

www. mathworks. com

Simulink 自动地对长的文本行进行折绕.

3.3　创建模板模块动态对话框

Simulink 允许用户创建根据用户的输入而改变的模板模块对话框，模板对话框改变特性包括：

(1) 参数控制的可视性

改变一个参数可引起对另一参数出现或不出现的控制. 控制的出现和消失，分别扩展和收缩对话框.

(2) 参数控制的激活状态

根据输入，改变参数可引起其它参数控制的激活或停止状态. Simulink 使非激活状态的控制变为灰色，表明其明显的非激活状态.

(3) 参数值

改变一个参数，可以引起相关参数设置为适当的值.

创建一个动态模板对话框，必须结合 Simulink 的 set_param 命令来使用模板编辑器. 首先，无论是静态的还是动态的参数，使用模板编辑器定义所有对话参数；其次，在 MATLAB 命令行中，使用 Simulink 的 set_param 命令来指定回调函数，这些函数是来定义用户输入对话响应的；最后，保存包含模板子系统的模型或库来完成动态模板对话框的创建.

3.3.1　设置模板模块的对话参数

Simulink 定义一套模板模块参数来定义模板模块对话框的当前状态. 用户可以使用模板编辑器来指定和设置其中的许多参数. Simulink 的 get-param 和 set-param 命令，可允许用户检查和设置模板对话参数. set-param 命令允许用户设置参数，因此，当对话框打开时，可以改变其外观，这样就可以产生动态模板对话框. 例如，用户可以在 MATLAB 命令行中，使用 set-param 命令，指定当改变用户自定义参数值时，要调用的回调函数. 而回调函数也可以使用 set-param 命令改变模板对话框预先定义参数的值及其状态，如：隐藏、显示、激活、非激活用户自定义参数控制. 这意味着用户可用 M 文件完成复杂的回调函数.

3.3.2　预定义模板对话参数

Simulink 将下列预定义参数与模板对话框相联系.

(1) 参数：MaskCallbacks

该参数的值是一字符串单元数组，指定对话用户自定义参数控制的表达式. 第一个单元定义第一个参数控制的回调函数，第二个单元定义第二个参数控制的回调函数，依次类推. 回调函数可以是任意合法 MATLAB 表达式，包含调用 M 文件命令的表达式. 为模板对话框设置回调函数最简单的方法，是首先在模型或库窗口中选择相应的模板对话框，然后在 MATLAB 命令行中使用 set-param 命令.

例 3.10 下面的代码是使用 set-param 命令设置 MaskCallbacks 参数的例子.

```
set_param(gcb,'MaskCallbacks',{'parm1_callback','','parm3_callback'});
```

该命令为当前所选模块的模板对话框的第一、第三个参数定义回调函数. 只要保存包含模板模块的模型或库,就将该设置保存起来.

(2) 参数:MaskDescription

该参数的值是指定模块描述的字符串. 用户可以通过设置该参数来动态改变模板模块的描述.

(3) 参数:MaskEnables

该参数的值是一个字符串单元数组,这些字符串是用来定义对话框中用户自定义参数控制激活状态的. 第一个单元定义第一个参数的控制的激活状态,第二个单元定义第二参数,依次类推. 值为'on',表示对于用户的输入,相应的控制是激活的;值为'off',表示对于用户的输入,相应的控制是非激活的. 可以通过在回调函数中设置该参数来动态地激活或关闭用户的输入.

例 3.11 在回调函数中输入下面的命令:

```
set_param(gcb,'MaskEnables',{'off','on','off'});
```

该命令将关闭当前模板模块对话的第一个和第三个控制. Simulink 将控制着灰色,以表明其非激活状态.

(4) 参数:MaskPrompts

该参数值是用来指定用户自定义参数提示的字符串单元数组. 第一个单元定义第一个参数的提示,第二个单元是第二个参数的提示,依次类推.

(5) 参数:MaskType

该参数的值是与对话框相关模块的模板类型.

(6) 参数:MaskValues

该参数的值是指定对话框的用户自定义参数值的字符串单元数组. 第一个单元定义第一个参数的值,第二个单元定义第二个参数的值,依次类推.

(7) 参数:MaskVisibilities

该参数值是一个字符串单元数组,该字符串用来指定用户自定义对话控制的可视性. 第一个单元定义第一个参数控制的可视性,第二个单元定义第二参数控制的可视性,依次类推. 值为'on',表示相应的控制是可视的;值为'off',相应的控制是隐藏的. 可以在控制的回调函数中,通过设置该参数来动态显示或隐藏用户自定义参数控制.

例 3.12 在回调函数中输入如下命令:

```
set_param(gcb,'MaskVisibilities',{'on','off','on'});
```

该命令将隐藏当前所选模块的第二个用户自定义模板参数. Simulink 可以扩展或收缩对话框,分别显示或隐藏控制.

3.4 条件执行子系统(conditionally executed subsystem)

条件执行子系统的执行取决于某一输入信号的值. 控制一个子系统是否执行的信号被称为控制信号. 控制信号从控制输入端口进入子系统模块.

如果在创建某些复杂的模型时，它们包含的一些组件的执行取决于另外一些组件，那么条件执行的子系统此时就会非常有用.

Simulink 支持三类条件执行的子系统：

1) 激活(enabled)子系统，当控制信号为正时激活子系统执行. 激活子系统在控制信号从负变为正(过零)时开始执行，并且只要控制信号保持为正它就一直运行.

2) 触发(triggered)子系统，每当"触发事件"发生时，它就执行一次. 触发事件可以发生在触发信号的上升沿或下降沿，信号可以是连续信号也可以是离散信号.

3) 可触发与激活子系统，当触发事件发生时，如果激活控制信号为正值，它就执行.

3.4.1 激活子系统

激活子系统是当控制信号为正值时的每一仿真步都执行的子系统.

激活子系统有一个惟一的控制信号输入端口，它可以是标量或向量输入端口：

1) 如果输入是标量，当输入值大于零时子系统就执行.

2) 如果输入是向量，则当输入向量的任一元素的值大于零时，子系统就执行.

例如，如果控制输入信号是一正弦波，则子系统在激活和非激活两种状态下交替.

Simulink 使用过零点的斜率来确定是否发生了激活事件. 如果信号穿越了零点，且斜率为正，子系统将变为运行状态；如果信号穿越零点时，斜率为负，子系统将变为非激活状态.

3.4.1.1 创建激活子系统

可以通过从 Signals & Systems 库中拷贝一个 Enable 模块到子系统中来创建一个激活子系统. Simulink 在 Subsystem 模块图标中增加一个激活符号和一个激活控制输入端口，如图 3.25 所示.

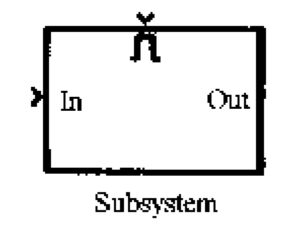


图 3.25 激活子系统图标

(1) 当子系统变为非激活时设置输出值

尽管激活子系统在非激活状态时不执行，但它们的输出信号仍然可以提供给另外的模块. 当激活子系统处于非激活状态时，可以选择保持它的输出信号为在它变为非激活状态前的值，或重置为初始值.

打开子系统中每一个 Outport 模块对话框，如图 3.26 所示，并且为 Output when disabled 参数选择一个选项值：

1) 选择 held，以使输出保持为其最近的值.

2) 选择 reset，以使输出重置为初始状态. 设置 Initial output 为输出的初始值.

(2) 当子系统重新变为激活时设置状态

当激活子系统执行时，可选择是保持子系统的状态为它们先前的值还是重置它们为它们的初始状态.

要设置当子系统重新变为激活时的状态，打开子系统中 Enable 模块对话框，如图 3.27 所示，并且为 States when enabling 参数选择一个选项值：

1) 选择 held，以使状态保持为它们最近的值.

2) 选择 reset，以使状态重置为它们的初始值.

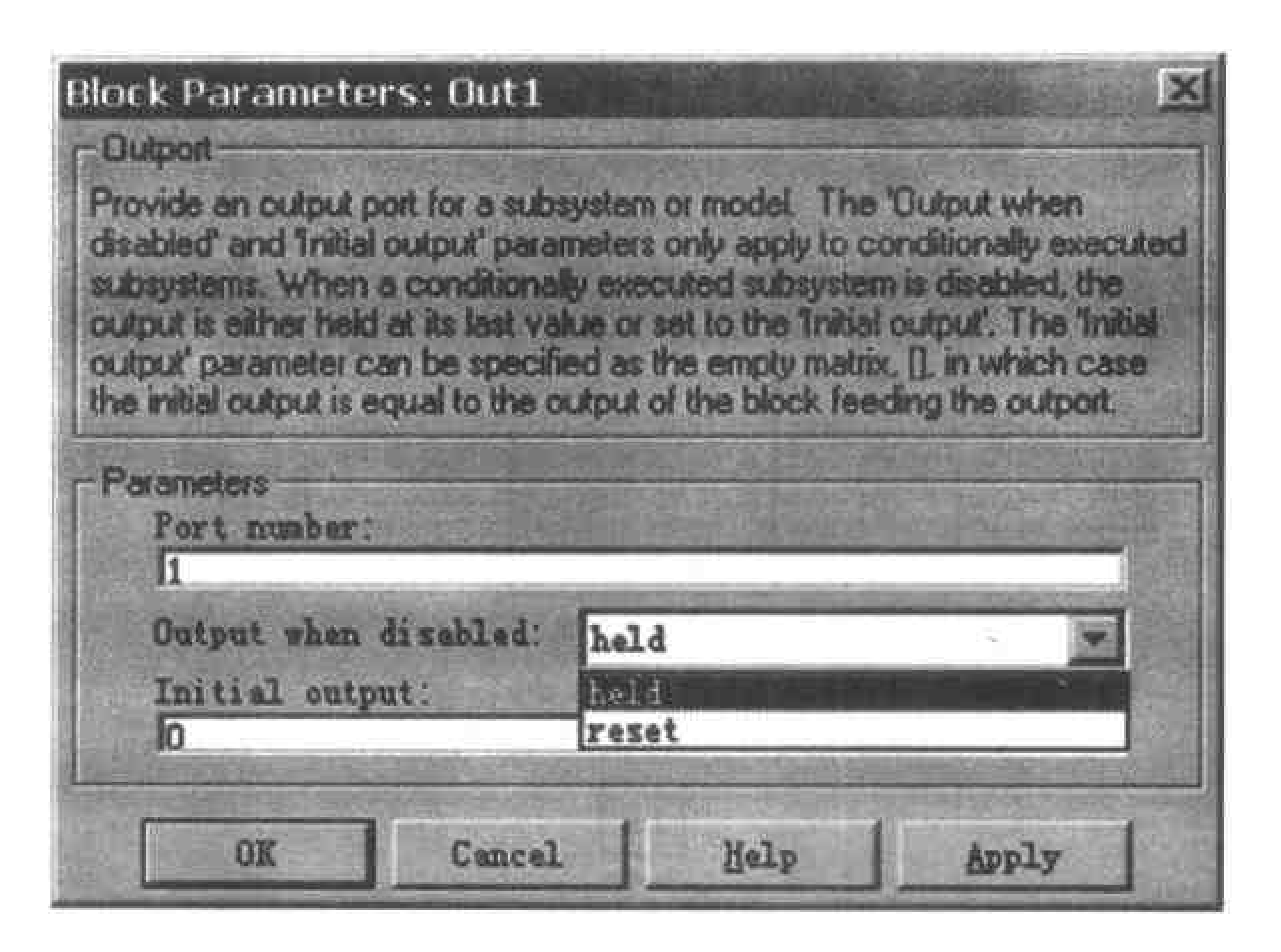


图 3.26 输出端口模块参数对话框

(3) 输出激活控制信号

通过子系统的 Enable 模块对话框中的一个选项可以输出激活控制信号. 要输出控制信号,选择 Show output port 核选框,如图 3.27 所示.

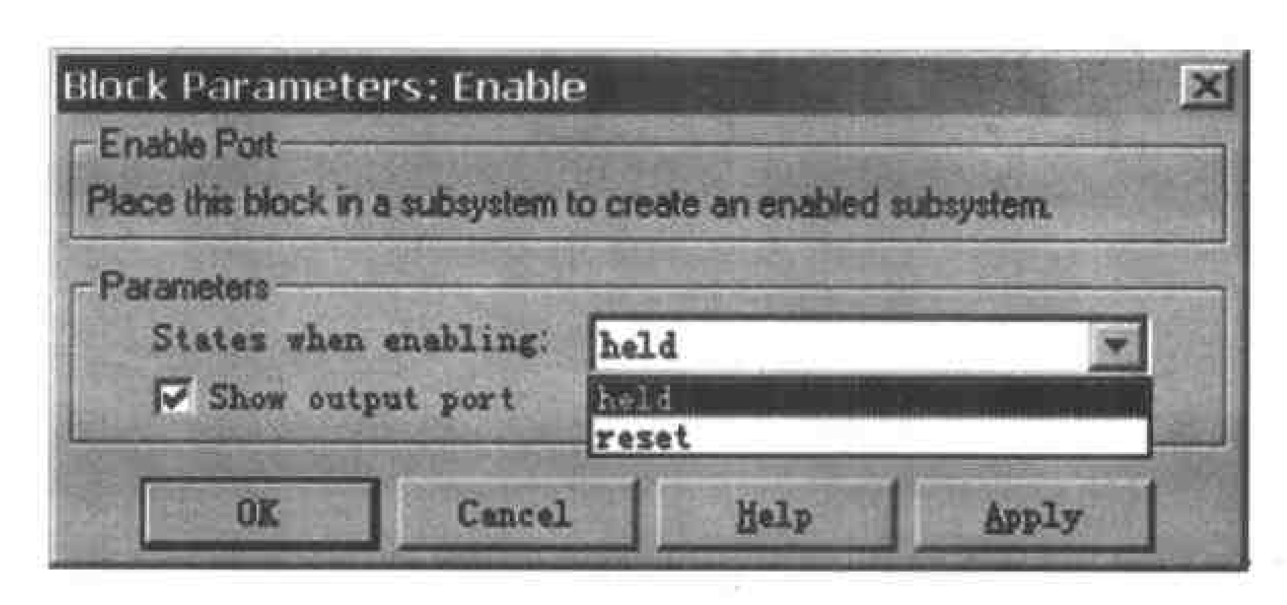


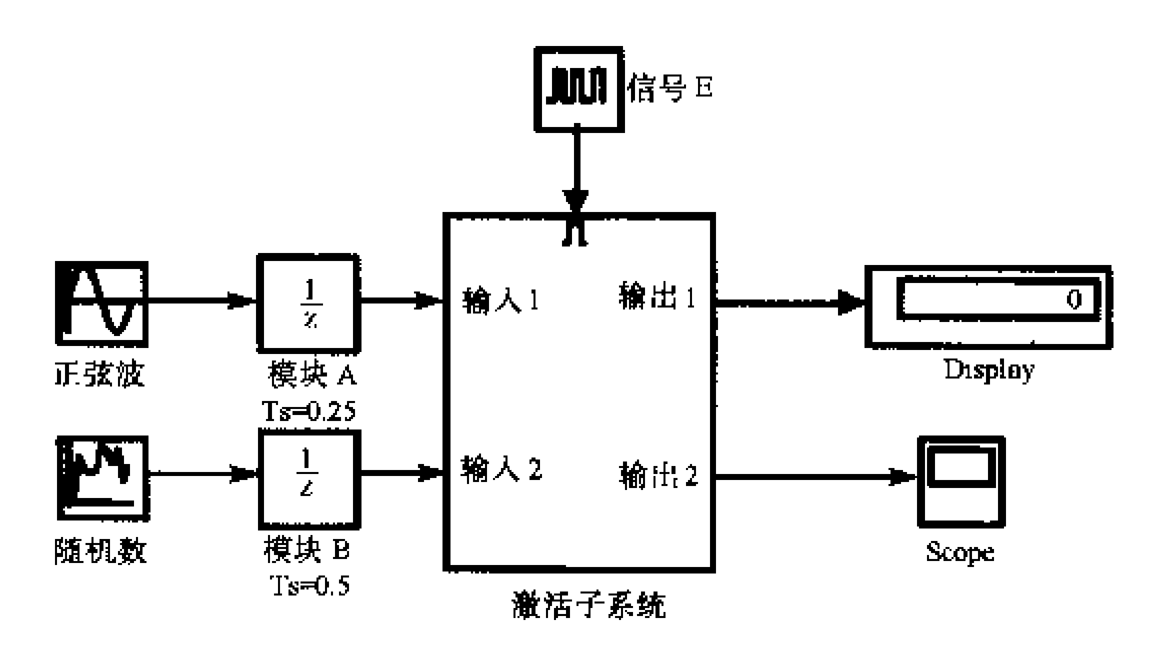
图 3.27 激活(Enable)模块参数对话框

通过这一特性可以传递控制信号到激活子系统,它们对于激活子系统中取决于包含在控制信号内的值的逻辑运算非常有用.

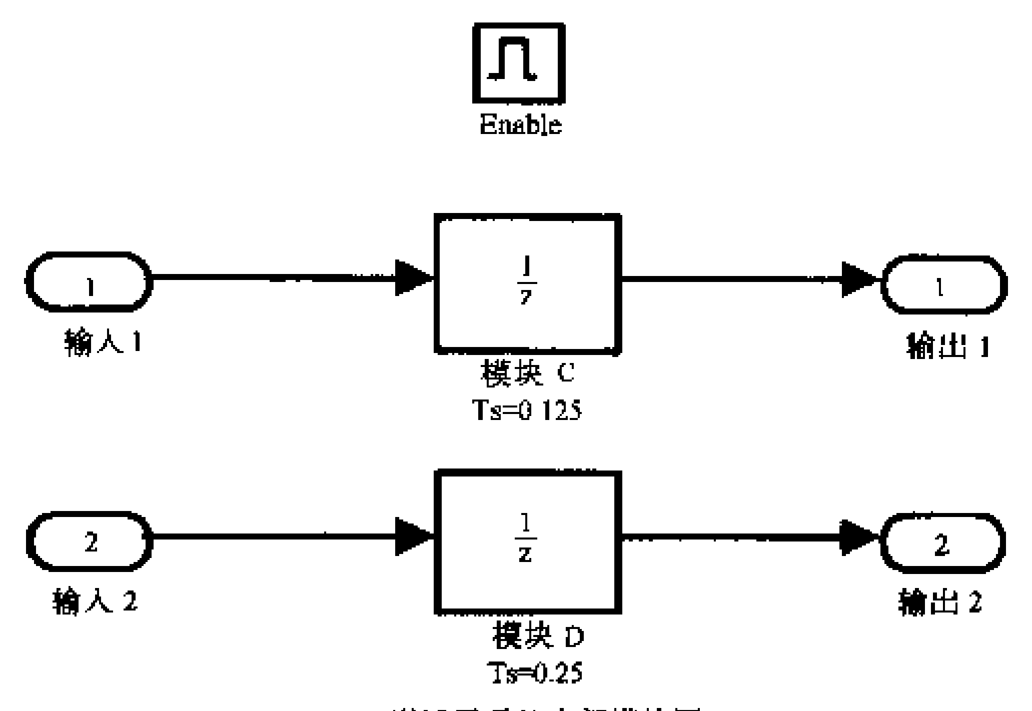
3.4.1.2 激活子系统能够包含的模块

激活子系统能够包含任何模块,可以是连续的也可以是离散的. 激活子系统中的离散模块只有当子系统执行时,且仅当它们的采样时间与仿真的采样时间同步时,它们才执行. 激活子系统和模型使用一个共同的时钟.

例 3.13 下面的系统包含有四个离散的模块和一个控制信号,其模型及子系统如图 3.28 所示. 离散的模块分别是:



(a) 建立包括激活子系统的模型



(b) 激活子系统内部模块图

图 3.28 激活子系统模型

模块 A，采样时间 Ts＝0.25 秒.

模块 B，采样时间 Ts＝0.5 秒.

模块 C，在 Enabled 子系统内，采样时间 Ts＝0.125 秒.

模块 D，在 Enabled 子系统内，采样时间 Ts＝0.25 秒.

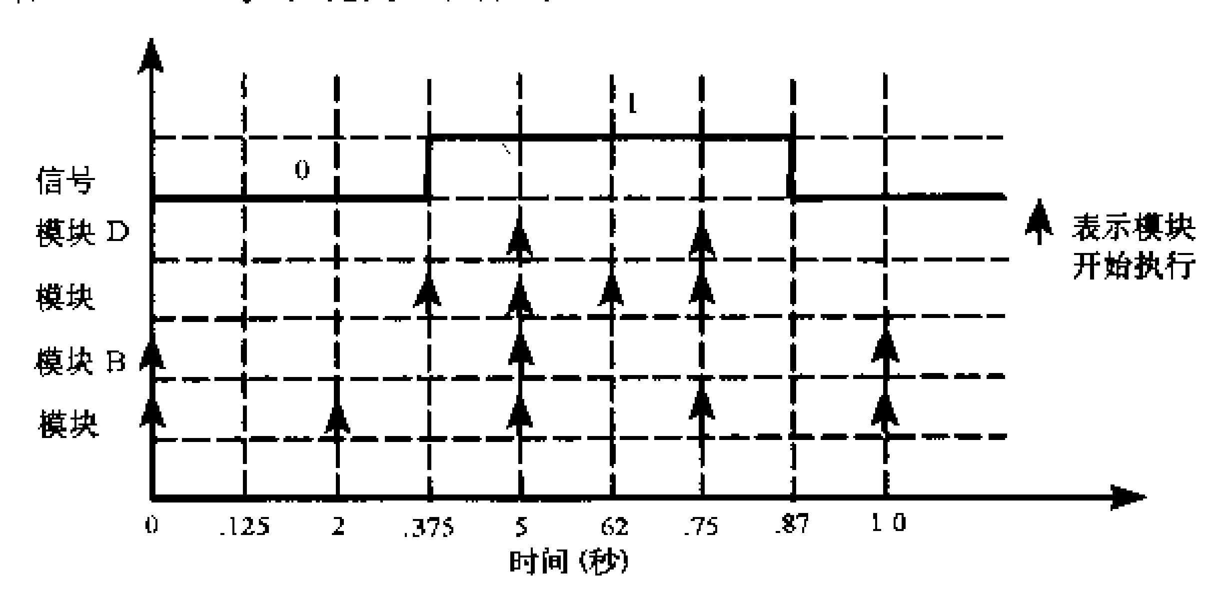


图 3.29 各离散模块执行时间

控制信号由 Pulse Generator 模块产生，被标为信号 E，在 0.375 秒时，它的值从 0 变到 1，在 0.875 秒时，它的值从 1 变为 0.

图 3.29 显示了各离散模块执行的时间.

模块 A 和模块 B 的执行与激活信号无关，因为它们不是激活子系统的一部分. 当激活信号变为正时，模块 C 和模块 D 以给它们指定的采样速率执行，直到激活信号又变为负. 注意到模块 C 在 0.875 秒时，激活信号变为零，它不运行.

3.4.2 触发子系统

每当触发事件发生时触发子系统就执行.

触发子系统有一个惟一的控制信号输入端口，被称为触发输入，它决定子系统是否执行. 可以从下面三种触发事件中选择一种以强制一个触发子系统开始执行：

1) 上升沿触发器. 当控制信号，或控制信号向量的任何一个元素的值从 0 变为正时（或初始值负时的 0）.

2) 下降沿触发器. 当控制信号，或控制信号向量的任何一个元素的值从 0 变为负时（或初始值为正时的 0）.

3) 边沿触发器. 当控制信号，或控制信号向量的任何一个元素的值从 0 变为非 0 时.

图 3.30 显示了一个给定控制信号的上升沿(R)和下降沿(F)触发器发生的时刻.

图 3.31 是一个含触发子系统的例子.

在该例中，可触发子系统由脉冲触发控制信号的上升沿触发.

(1) 创建一个触发子系统

可以通过从 Signals & Systems 库中拷贝 Trigger 模块到一个子系统来创建一个可触发子系统. Simulink 在 Subsystem 模块图标中增加一个触发符号和一个触发控制信号输入端口.

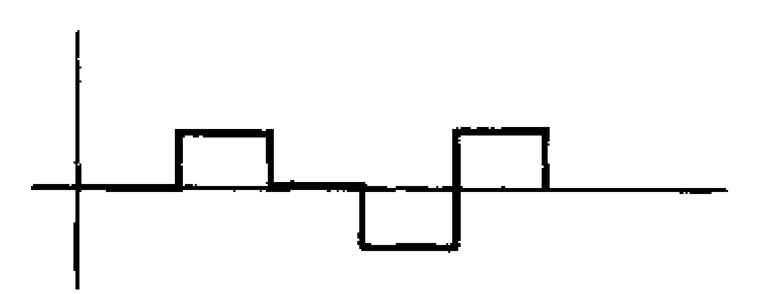


图 3.30 触发器控制信号的上升沿(R)和下降沿(F)

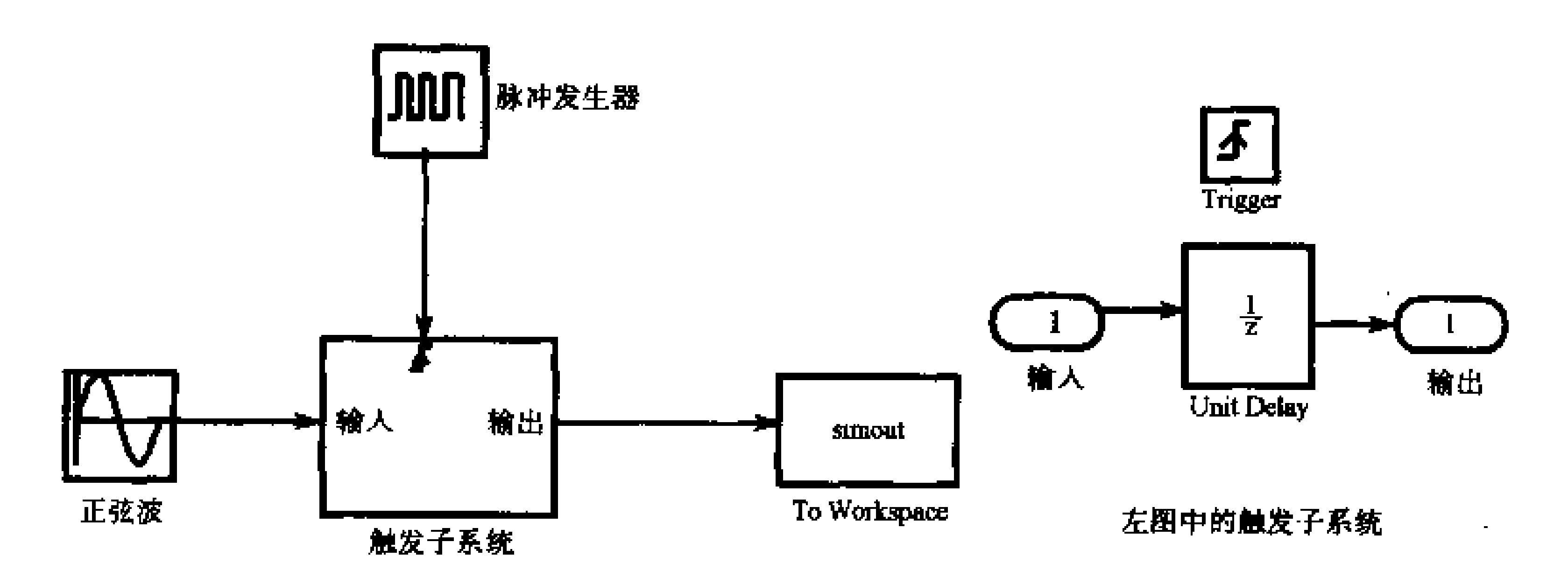


图 3.31 含触发子系统模型

要选择触发类型，打开 Trigger 模块对话框，如图 3.32 所示，并为 Trigger type 参数选择一个选项：

1) rising(上升触发)：每当触发信号穿越零变为正值时，产生触发事件.

2) falling(下降触发)：每当触发信号穿越零变为负值时，产生触发事件.

3) either(上升和下降都触发)：每当触发信号穿越零时，产生触发事件.

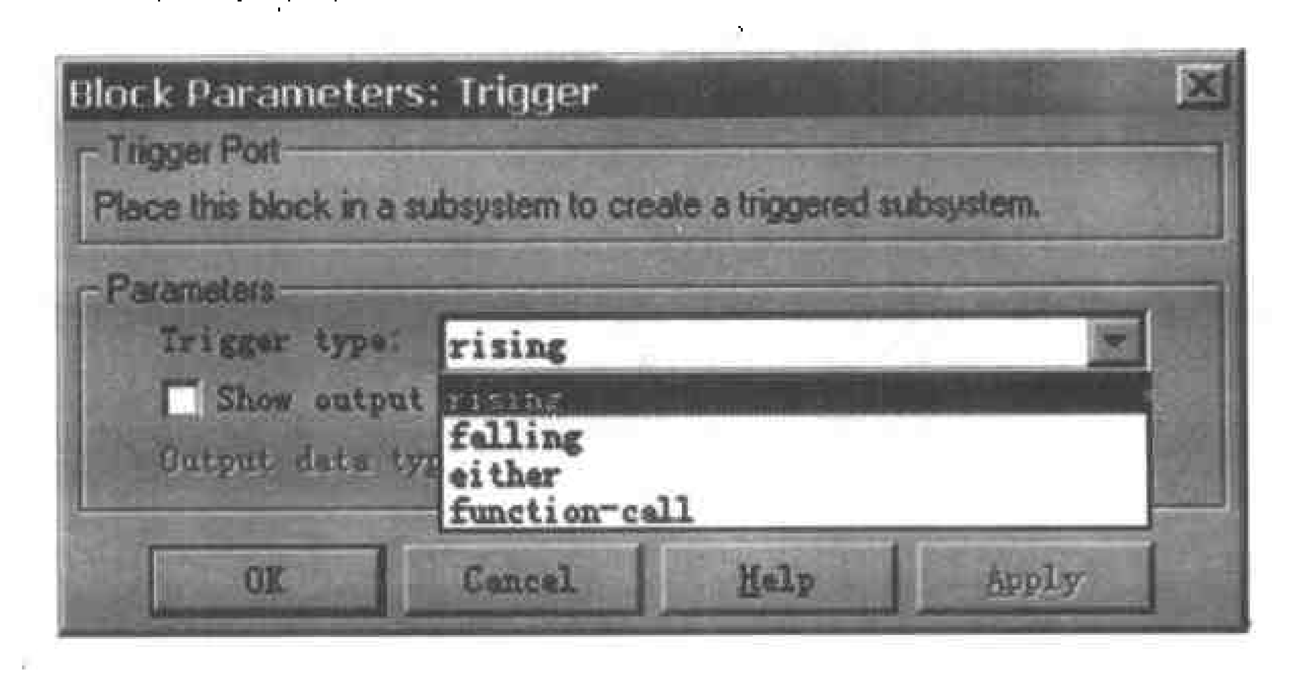


图 3.32　Trigger 模块参数对话框

Simulink 在 Trigger 和 Subsystem 模块中使用不同的符号来表示上升沿和下降沿触发器(或两者). 图 3.33 显示了 Subsystem 模块中用到的三种触发器符号.

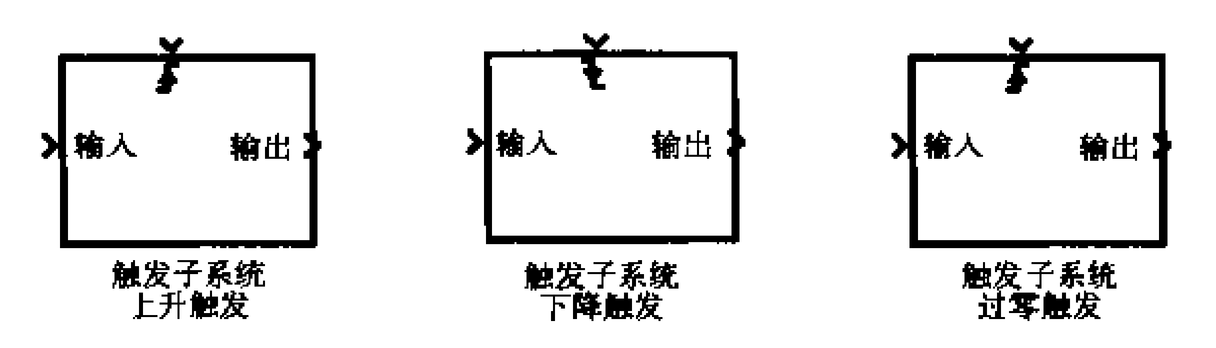


图 3.33　三种触发类型及其符号

1) 触发事件之间的输出和状态. 与激活子系统不同，触发子系统通常在触发事件之间的输出保持为最近的值. 同样，触发子系统在被触发时不能重置它们的状态；任何离散模块的状态在触发事件之间保持不变.

2) 输出触发控制信号. 在 Trigger 模块参数对话框中有一个选项，通过选取它可以输出触发控制信号. 要输出控制信号，选择 Show output port 复选框，如图 3.34 所示.

在 Output data type 域中可以指定输出信号的数据类型为：auto，int8 或 double. auto 选项使输出信号的类型设置为与信号相连端口的数据类型.

(2) 函数调用子系统(Function-Call Subsystems)

可以创建一个触发子系统，它的执行取决于一个 S 函数的内部逻辑，而不是信号的值. 这些子系统被称为函数调用子系统.

(3) 可触发子系统能够包含的模块

触发系统在仿真期间只在特定的情况执行. 这样，在触发子系统中可以使用的模块只包括：

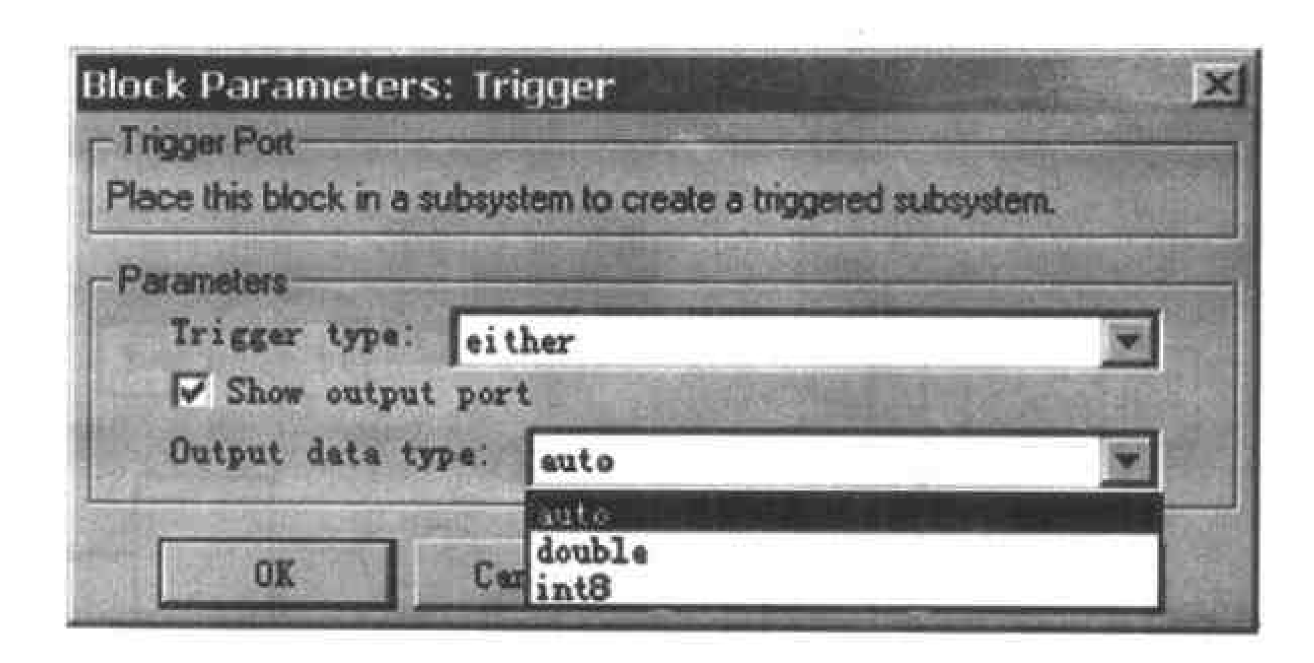


图 3.34 选择 Show output port 复选框

1) 能够继承采样时间的那些模块. 如 Logical Operator 模块或 Gain 模块.

2) 采样时间设为－1 的离散模块. 采样时间设为－1，表示它继承驱动模块的采样时间.

3.4.3 触发与激活子系统

第三种条件执行的子系统，是前两种条件执行子系统的组合. 这种被称为触发与激活子系统的行为是激活子系统和可触发子系统的组合，其流程如图 3.35 所示.

触发与激活的子系统包含一个激活输入端口和一个触发输入端口，当触发事件发生时，Simulink 检查激活输入端口以计算激活控制信号的值，如果它的值大于 0，Simulink 执行子系统. 如果两个端口输入的都是向量，如果每一向量至少有一个元素为非 0 值，则子系统执行.

当触发事件发生时，子系统在每一时间步执行一次.

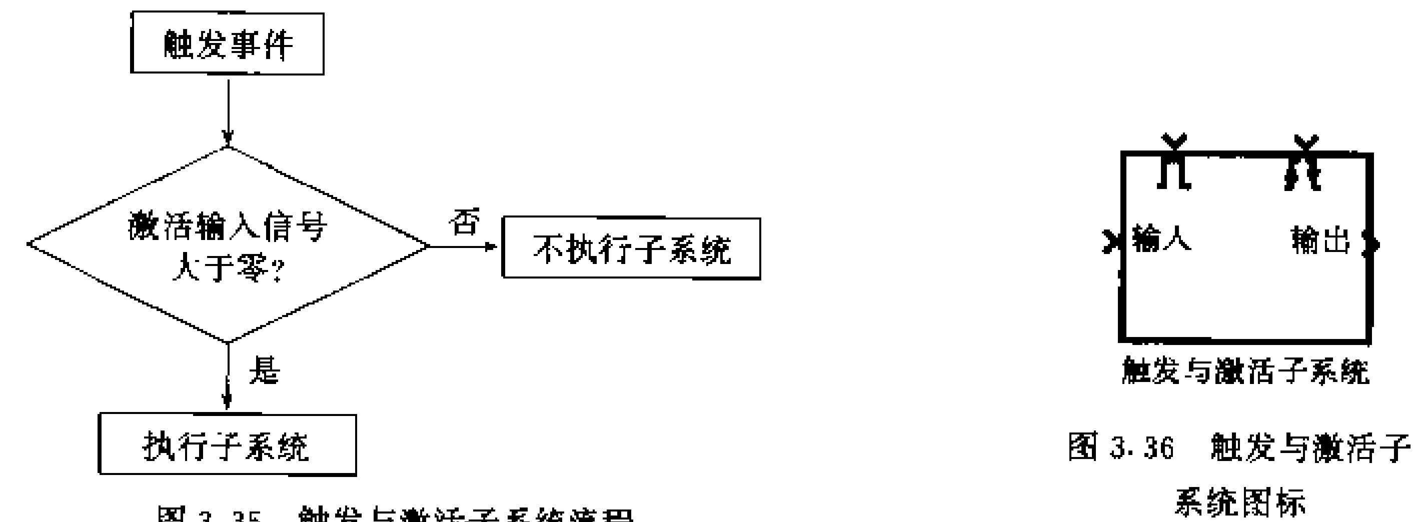


图 3.35 触发与激活子系统流程

图 3.36 触发与激活子系统图标

(1) 创建触发与激活子系统

从 Signals & Systems 库中，拷贝 Enable 和 Trigger 模块到一个已存在的子系统中，创建一个触发与激活子系统. Simulink 在 Subsystem 模块图标中增加激活和触发符号及激活和触发控制信号输入端口，如图 3.36 所示.

可以设置当触发与激活子系统为非激活时的输出值，就像一个激活子系统那样；也可以指定当子系统重新变为激活时的状态值.

另外，可以分别设置 Enable 和 Trigger 模块的参数值. 设置的过程与前面各模块介绍的方法相同.

(2) 采样触发与激活子系统

图 3.37 所示模型是一个触发与激活子系统的简单例子.

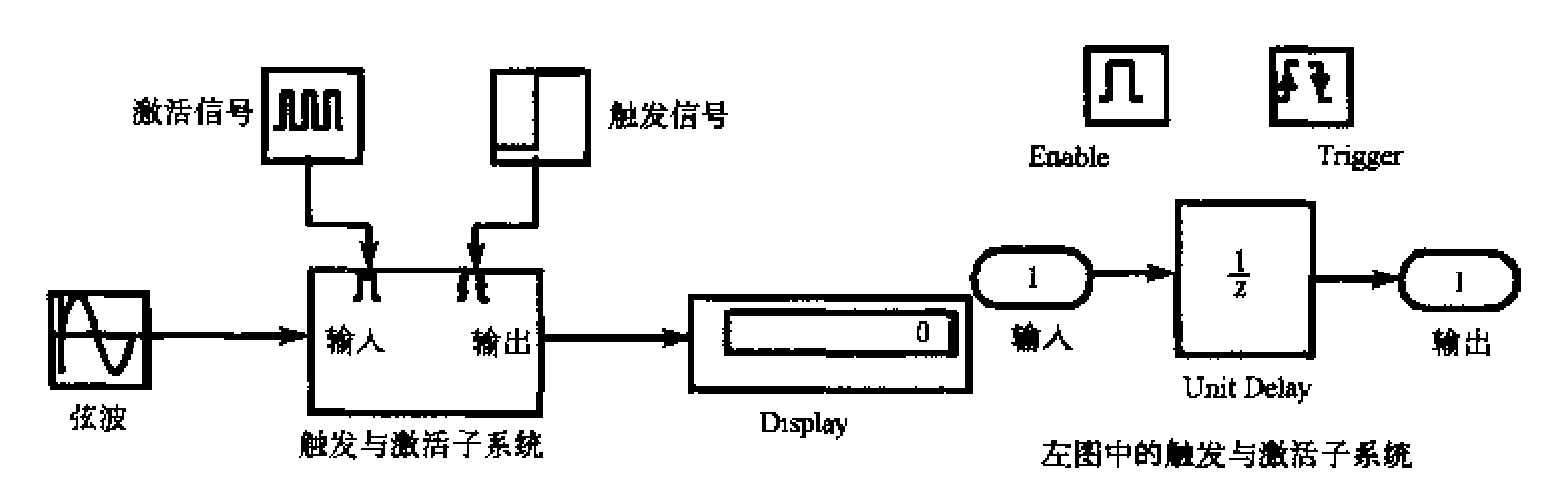


图 3.37　采样触发与激活子系统

(3) 创建交替执行子系统

将条件执行子系统与合并(Merge)模块相结合，可以创建一些依赖模型当前状态的交替执行的子系统. 图 3.38 就是一个使用两个激活(enabled)模块和一个合并模块的模型，这是一个变换器模型，将交流(AC)转换为直流脉冲(DC).

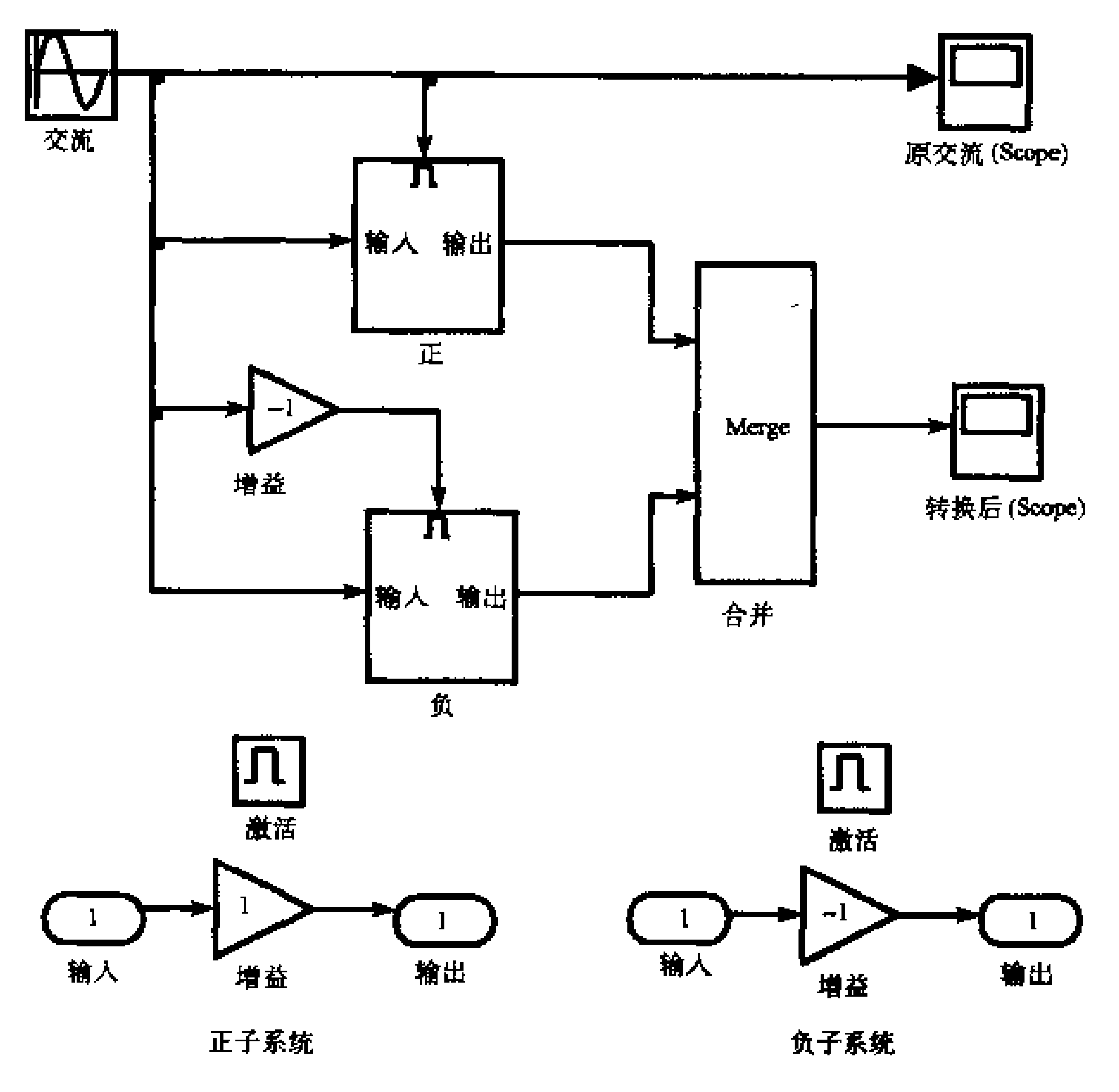


图 3.38　交流(AC)转换为直流脉冲(DC)模型

该模型系统中，当交流信号为正时，标有“正”的模块被激活，交流信号直通输出；当交流信号为负时，标有“负”的模块被激活，交流信号被倒置；合并模块将当前激活模块的输出传送至转换后 Scope 模块. 原始交流信号与转换后结果如图 3.39 所示.

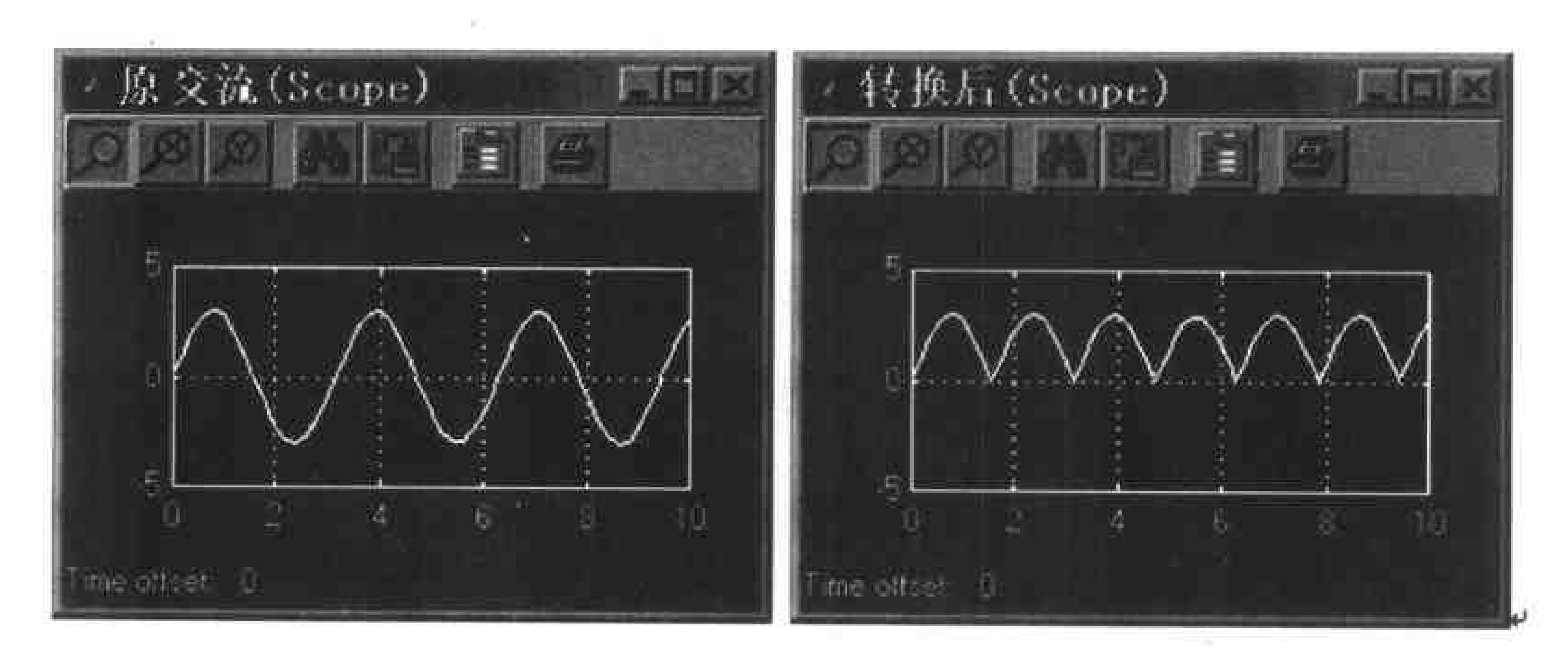


图 3.39　原交流与转换后结果比较

第四章　运行 Simulink 仿真

Simulink 的仿真运行可以通过菜单进行，也可在 MATLAB 的命令窗口中输入命令. 许多用户在开发和修改模型阶段喜欢用菜单命令，而当“批量”运行仿真时，会选择在 MATLAB 命令窗口中输入命令.

4.1　使用菜单命令运行仿真

通过菜单命令运行仿真非常简单，并且交互性比较好. 这些命令使得用户不用记住命令语法就可以选择 ODE 求解器或者定义仿真参数. 菜单命令的一个重要优点是当运行仿真时可以交互地做如下工作：

1）可以改变许多仿真参数；包括停止时间、求解器和最大步长.

2）可以改变求解器.

3）可以同时运行其它的仿真.

4）可以点击某条连线，在一个孤立的（没有相连）Scope 或 Display 模块中查看该条连线上传输的信号.

5）可以改动模块的参数，只要不引起以下变化：

- 状态、输入和输出的数目；
- 采样时间；
- 过零点的数目；
- 任何模块参数的向量长度；
- 内部模块工作向量的长度.

在仿真期间，不能改变模型的结构，诸如增加、删除连线或模块. 如果要对模型改动，必须终止仿真，修改完以后再运行仿真来观察结果的改变.

4.1.1　设置仿真参数和选择求解器

通过选择Simulation 菜单下的 Parameters 菜单项，Simulink 显示 Simulation Parameters 对话框，如图 4.1 所示，用来设置仿真参数和选择求解器. Simulink 显示的仿真参数(simulation parameters)对话框，其中有三个页面管理如下的这些仿真参数：

1）在 Solver 页面，可以设置开始和停止时间，选择求解器和指定求解器(solver)的参数，另外还可以选择一些输出选项.

2）Workspace I/O 页面，管理对 MATLAB 工作空间的输入和输出.

3）在 Diagnostics 页面，可以选择在仿真期间显示的警告信息的层次.

可以使用有效的 MATLAB 表达式设定参数，这些表达式通常由常量、工作空间变量名、MATLAB 函数和数学运算符组成.

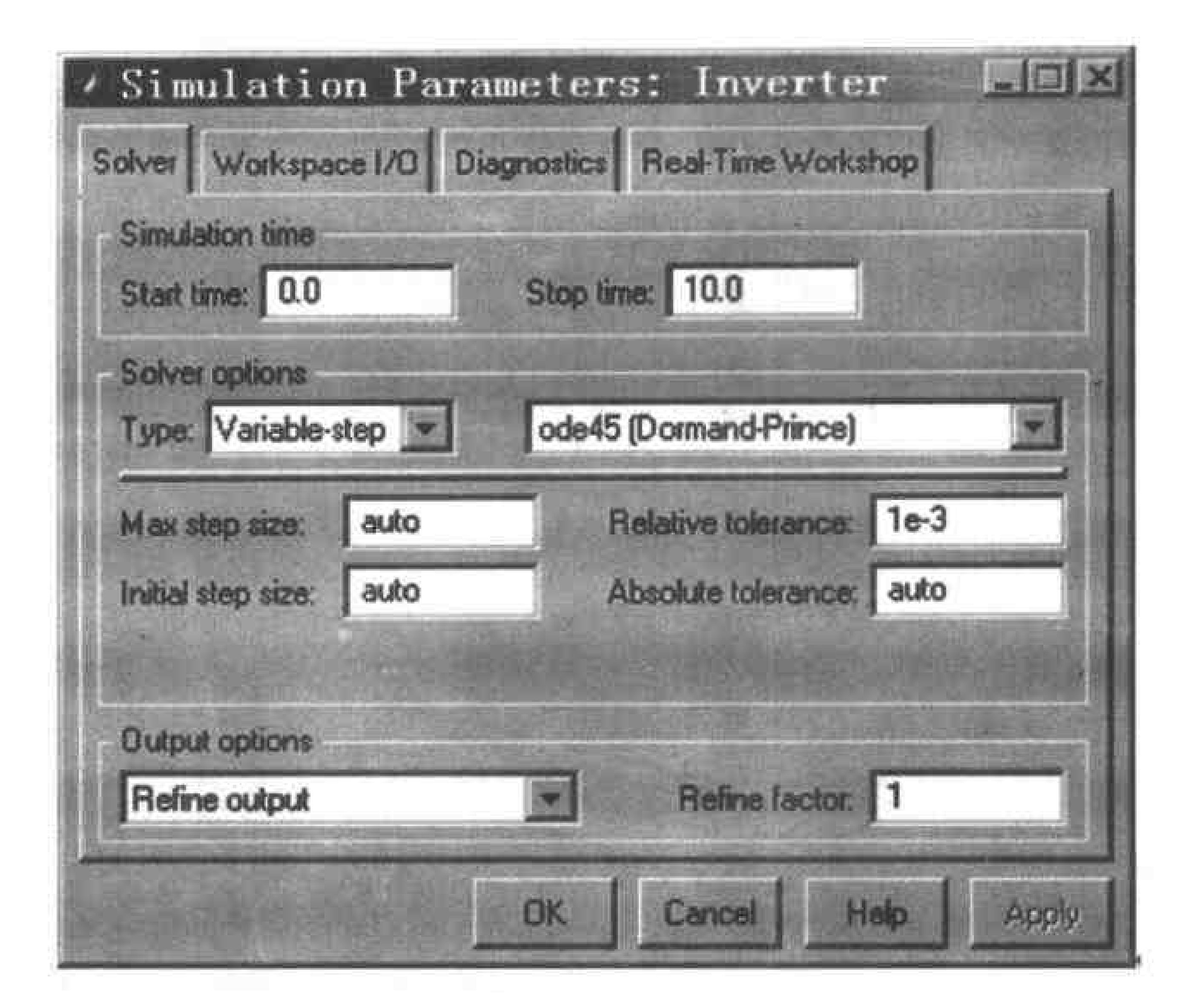


图 4.1 仿真参数对话框

4.1.2 应用仿真参数

在设置好仿真参数并选择好求解器后，就可以在模型中使用这些设定. 点击对话框底部的 Apply 按钮，在模型中使用这些新设置定的参数. 要在使用参数的同时关闭对话框，点击 OK 按钮.

4.1.3 开始仿真

设定并应用仿真参数和求解器以后，就可以运行仿真了. 选择Simulation 菜单下的 Start 菜单项，或者使用键盘的快捷键 Ctrl+T 运行仿真. 在选择 Start 菜单项以后，该菜单项将变成 Stop. 在仿真结束的时候，计算机会发出蜂鸣信号.

Simulink 的初学者常犯的一个错误就是，在 Simulink 的模块库是活动窗口时运行仿真，在运行仿真之前必须确保模型窗口是活动窗口.

要终止仿真的运行，选择Simulation 菜单下的 Stop 菜单项，它的键盘快捷键与开始仿真的键盘快捷键相同(Ctrl+T).

选择Simulation 菜单下的Pause 菜单项，可以挂起一个运行着的仿真. 当选择Pause 时，该菜单项变成Continue. 可以选择Continue 菜单项继续仿真的运行.

如果模型中包含有写数据到文件或工作空间的模块，或者在 Simulation Parameters 对话框中选择了输出选项，在仿真被终止或挂起时 Simulink 将会写这些数据.

4.1.4 仿真诊断(Simulation Diagnostics)对话框

如果在仿真期间出现错误,Simulink 就会停止仿真,并在仿真诊断对话框中显示错误信息,如图 4.2 所示.

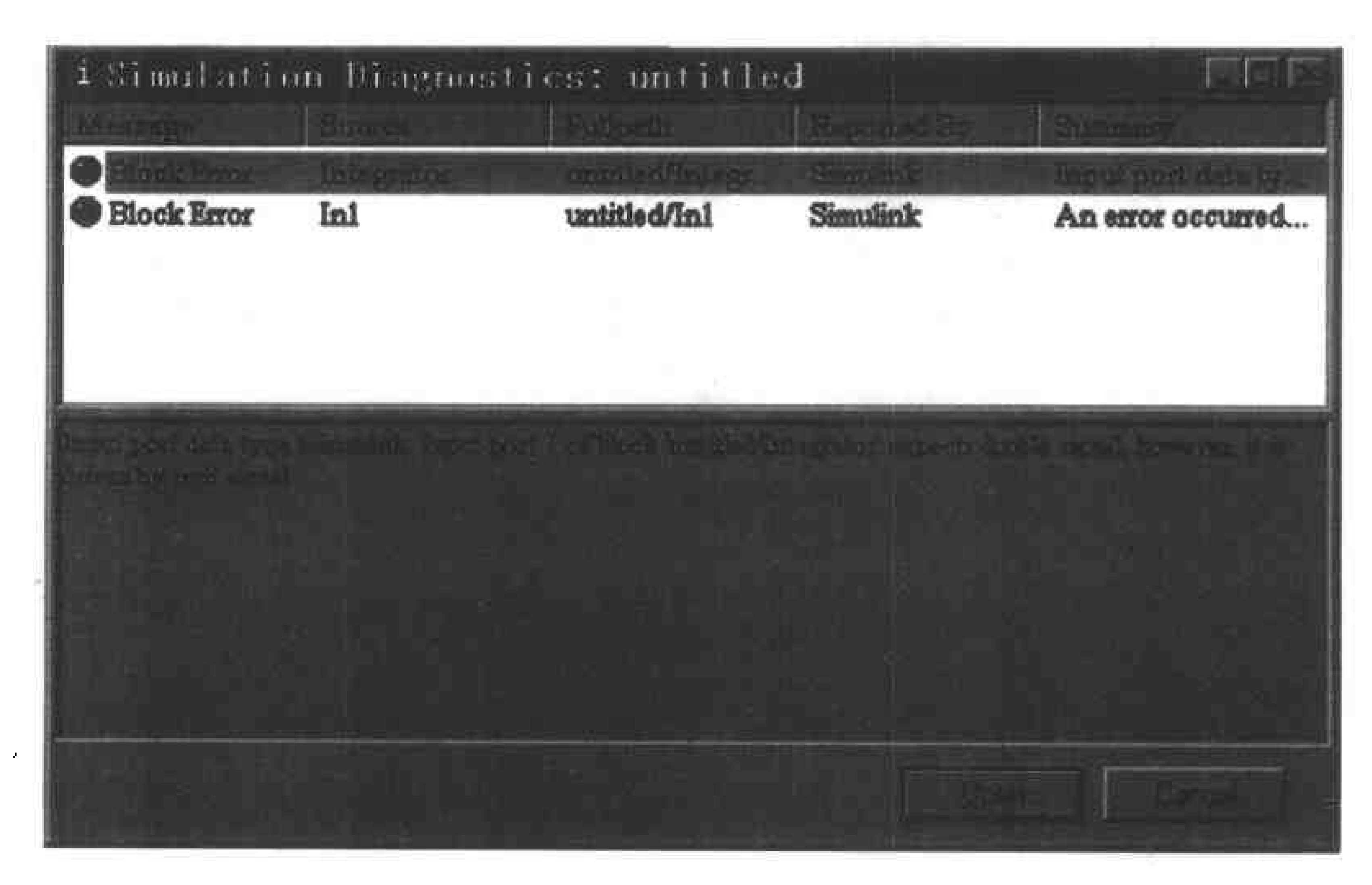


图 4.2 仿真诊断对话框

仿真诊断对话框包括两个窗口,上部的窗口显示如下一些信息的列组成:

1) Message:错误信息类型,如模块错误、警告、日志.

2) Source:引起错误的模型元素名字,如模块名字.

3) Fullpath:引起错误的元素路径.

4) Reported by:报告错误的部件,如 Simulink,Stateflow, Real-Time Workshop 等等.

5) Summary:错误信息缩短以适合该列显示.

仿真诊断对话框下部的窗口,在其最上部,最初包含第一个错误信息的全部内容.可以通过双击上部窗口中的其它错误信息,也可以双击错误源的名字,或按对话框中的 Open 按钮,来显示其它错误源.

在显示仿真诊断对话框时,必要时 Simulink 也会打开包括错误源的模型框图,并突出显示其错误源.

4.2 仿真参数对话框

仿真参数既可以在 Simulation Parameters 对话框(如图 4.1 所示)中设定,又可使用 sim 和 simset 命令设定.

表 4.1 总结了仿真参数对话框的每一个页面中都显示的四个按钮的作用.

表 4.1　仿真参数对话框中的按钮

按　钮	动　作
OK	使用新设定的参数值并关闭对话框. 在仿真运行时,参数值会被立即应用到仿真过程中
Cancel	使参数值变为对话框最初被打开时的值并重新使用这些参数值
Help	显示关于该对话框的帮助
Apply	使用新设定的参数值但不关闭对话框. 在仿真运行期间,参数值会被立即应用到仿真过程中

下面分别介绍仿真参数对话框的每个页面.

4.2.1　Solver 页

第一次从 Simulation 菜单下选择 Parameters 菜单项,或者选择仿真参数对话框的 Solver 标签(tab),会显示 Solver 页面,如图 4.1 所示.

在 Solver 页中可以进行以下工作:

1) 设定仿真的开始和停止时间.

2) 选择求解器,并为其指定参数.

3) 选择输出选项.

4.2.1.1　仿真时间(Simulation Time)

可以在 Start time 和 Stop time 文本框中输入新的值来改变仿真的起始时间和停止时间;缺省的开始时间为 0.0 秒,停止时间为 10.0 秒.

仿真时间和实际的时钟时间不是一回事. 例如,运行一个 10 秒钟的仿真通常要不了 10 秒钟的时间. 实际上运行一次仿真需要的时间要取决于多种因素,包括模型的复杂程度,求解器的步长和计算机的时钟频率等.

4.2.1.2　求解器

Simulink 模型的仿真包含一系列一般微分方程(ODEs)的数值积分,Simulink 为仿真提供了许多种计算这些方程的求解器. 由于不同动态系统的行为差异,所以在解决特定的问题时,某些求解器会比其它的更有效一些. 为了得到精确且迅速的计算结果,应当选择合适的求解器,并且设定合适的参数.

可以选择变步长和定步长的求解器. 变步长的求解器在仿真期间会改变其步长,它们提供误差控制和过零点检测. 定步长的求解器在仿真期间采用相同的步长,它们不提供误差控制也不定位过零点.

(1) 缺省求解器

如果不选择求解器,Simulink 将根据模型的状态自动选择一种求解器:

1) 如果模型是连续状态,将选择 ode45 求解器,ode45 是一个非常好的通用求解器. 然而,如果知道了系统是刚性的,且 ode45 没法提供可以接受的结果,可以用 ode15s 试一下. 对于刚性系统的定义,将在后面介绍.

2) 如果模型没有连续状态,Simulink 将使用被称作 discrete 的变步长求解器,并且

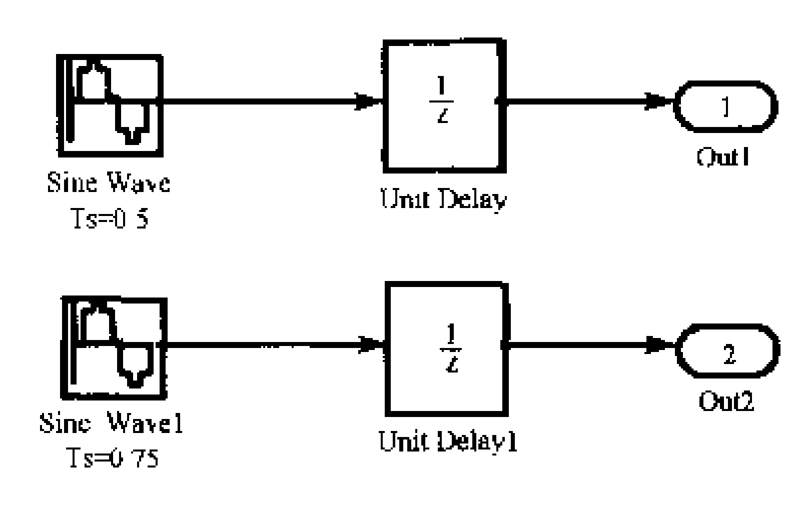


图 4.3 示例模型

显示一条信息以表明它使用的不是 ode45. Simulink 也提供名为 discrete 的定步长的求解器. 图 4.3 所示的模型显示了两种不同的 discrete 求解器之间的区别.

在上面的模型中,当采样时间为 0.5 和 0.75 秒时,基采样时间为 0.25 秒. 变步长和定步长 discrete 求解器之间的区别,在于它们产生的时间向量.

定步长 discrete 求解器产生的时间向量是:[0 0.25 0.5 0.75 1.0 1.25 ...]

例 4.1 设置定步长 discrete 求解器,并运行图 4.3 的模型后,输入命令:

```
tout'
ans =
  Columns 1 through 7
        0    0.2500    0.5000    0.7500    1.0000    1.2500    1.5000
  Columns 8 through 14
   1.7500    2.0000    2.2500    2.5000    2.7500    3.0000    3.2500
  Columns 15 through 21
   3.5000    3.7500    4.0000    4.2500    4.5000    4.7500    5.0000
  Columns 22 through 28
   5.2500    5.5000    5.7500    6.0000    6.2500    6.5000    6.7500
  Columns 29 through 35
   7.0000    7.2500    7.5000    7.7500    8.0000    8.2500    8.5000
  Columns 36 through 41
   8.7500    9.0000    9.2500    9.5000    9.7500   10.0000
```

变步长 discrete 求解器产生的时间向量是:[0 0.2 0.4 0.5 0.6 0.75 ...]

例 4.2 设置变步长 discrete 求解器,并运行图 4.3 的模型后,输入命令:

```
tout'
ans =
  Columns 1 through 7
        0    0.2000    0.4000    0.5000    0.6000    0.7500    0.8000
  Columns 8 through 14
   1.0000    1.2000    1.4000    1.5000    1.6000    1.8000    2.0000
  Columns 15 through 21
   2.2000    2.2500    2.4000    2.5000    2.6000    2.8000    3.0000
  Columns 22 through 28
   3.2000    3.4000    3.5000    3.6000    3.7500    3.8000    4.0000
  Columns 29 through 35
   4.2000    4.4000    4.5000    4.6000    4.8000    5.0000    5.2000
```

Columns 36 through 42

5.2500	5.4000	5.5000	5.6000	5.8000	6.0000	6.2000

Columns 43 through 49

6.4000	6.5000	6.6000	6.7500	6.8000	7.0000	7.2000

Columns 50 through 56

7.4000	7.5000	7.6000	7.8000	8.0000	8.2000	8.2500

Columns 57 through 63

8.4000	8.5000	8.6000	8.8000	9.0000	9.2000	9.4000

Columns 64 through 68

9.5000	9.6000	9.7500	9.8000	10.0000

定步长的 discrete 求解器的步长是基采样时间;而变步长的 discrete 求解器采用的是尽可能大的步长.

(2) 变步长(Variable-Step)求解器

可以选择的变步长求解器有:ode45,ode23,ode113,ode15s,ode23s 和 discrete. 缺省情况下,具有状态的系统用的是 ode45;没有状态的系统用的是 discrete.

1) ode45 基于显式 Runge-Kutta(4,5)公式,Dormand-Prince 对. 它是一个单步求解器(solver),也就是说它在计算 $y(t_n)$时,仅仅利用前一步的计算结果 $y(t_{n-1})$. 对于大多数问题,在第一次仿真时,可用 ode45 试一下.

2) ode23 是基于显式 Runge-Kutta(2,3),Bogacki 和 Shampine 对. 对于宽误差容限和存在轻微刚性的系统,它比 ode45 更有效一些. Ode23 也是单步求解器.

3) ode113 是变阶 Adams-Bashforth-Moulton PECE 求解器. 在误差容限比较严时,它比 ode45 更有效. Ode113 是一个多步求解器,即为了计算当前的结果 $y(t_n)$,不仅要知道前一步结果 $y(t_{n-1})$,还要知道前几步的结果 $y(t_{n-2})$,$y(t_{n-3})$,$y(t_{n-4})$,….

4) ode15s 是基于数值微分公式(NDFs)的变阶求解器. 它与后向微分公式 BDFs(也叫 Gear 方法)有联系,但比它更有效. ode15s 是一个多步求解器,如果认为一个问题是刚性的,或者在用 ode45s 时仿真失败或不够有效时,可以试试 ose15s.

5) ode23s 是基于一个 2 阶改进的 Rosenbrock 公式. 因为它是一个单步求解器,所以对于宽误差容限,它比 ode15s 更有效. 对于一些用 ode15s 不是很有效的刚性问题,可以用它解决.

6) ode23t 是使用"自由"内插式梯形规则来实现的. 如果问题是适度刚性,而且需要没有数字阻尼的结果,可采用该求解器.

7)ode23tb 是使用 TR BDF2 来实现的,即基于隐式 Runge-Kutta 公式,其第一级是梯形规则步长和第二级是二阶反向微分公式. 两级计算使用相同的迭代矩阵. 与 ode23s 相似,对于宽误差容限,它比 ode15s 更有效.

8) discrete(变步长)是 Simulink 在检测到模型中没有连续状态时所选择的一种求解器.

(3) 定步长(Fixed-Step)求解器

可以选择的定步长求解器有:ode5,ode4,ode3,ode2,ode1 和 discrete.

1) ode5 是 ode45 的一个定步长版本,基于 Dormand-Prince 公式.

2) ode4 是 RK4,基于四阶 Runge-Kutta 公式.

3) ode3 是 ode23 的定步长版本,基于 Bogacki-Shampine 公式.

4) ode2 是 Heun 方法,也叫作改进 Euler 公式.

5) ode1 是 Euler 方法.

6) discrete(定步长)是不执行积分的定步长求解器.它适用于没有状态的模型,以及对过零点检测和误差控制不重要的模型.

4.2.1.3 求解器选项

对于大多数问题来说,缺省的求解器参数能够提供比较精确和有效的计算结果.然而在有些情况下,调节参数可以提高仿真性能.可以在 Solver 页面中,通过改变所选求解器的参数值来调节求解器.

4.2.1.4 步长

对于变步长求解器,可以设定最大和建议的起始步长参数.缺省情况下这些参数被设为 auto,即意味着这些参数将被自动地设定.对于定步长,可以设定其步长.缺省时也为 auto.

(1) 最大步长(Maximum step size)

Max step size 参数控制最大的时间步长.缺省值由开始时间 t_{start} 和停止时间 t_{stop} 按式(4.1)决定.

$$h_{max} = \frac{t_{stop} - t_{start}}{50} \tag{4.1}$$

通常情况下,缺省的最大步长是足够的.如果对求解器可能失去某些重要的细节比较关心,可以改变参数来防止求解器采用太大的步长.如果仿真的时间跨度非常长,用缺省的步长计算得到最终结果可能要费很长时间.同样,如果模型中包含有周期性或近似周期性的细节并且知道其周期,可将最大的步长设为周期的几分之一(如 1/4).

一般来讲,要得到更多的输出点,是改变细化因子,而不是最大步长.

(2) 初始步长

缺省情况下,求解器通过检查开始时的状态导数来选择初始步长.如果第一步的步长选得太大,求解器可能跨过了重要的细节.起始步长参数值是建议第一步步长的,求解器将试着用这一步长,如果误差标准达不到要求,求解器将会减小步长.

4.2.1.5 误差容限

求解器使用标准的局部误差技术来监控每一步的误差.在每一步,求解器计算该步结束时的状态值,并且确定局部误差——这些状态值的估计误差,然后将局部误差与允许误差进行比较,这可用一个函数来实现,该函数是相对误差容限(rtol)和绝对误差容限(atol)的函数.如果对于任何一个状态,误差都大于允许值,求解器将减小步长重新计算.

1) 相对误差容限(Relative tolerance)是误差与状态值的比值.相对误差容限代表状态值的百分比.缺省情况下,1e-3 意味着计算结果的精确度将在 0.1%以内.

2) 绝对误差容限(Absolute tolerance)是误差的阈值.这一容限代表着当计算的状态

值趋于零时的允许误差.

第 i 步的状态误差 e_i 必须满足：$e_i \leqslant \max(\text{rtol} \times |x_i|, \text{atol}_i)$

图 4.4 说明了由相对误差容限和绝对误差容限决定的允许误差区域和状态.

如果指定缺省的 auto，Simulink 则设置每一状态的绝对误差容限初始为 1e-6. 随着仿真的运行，Simulink 为每个状态设置绝对误差容限和相对误差容限至最大值. 如果状态从 0 到 1，reltol 为 1e-3，那么，到仿真结束，abstol 也设置为 1e-3. 如果状态从 0 到 1000，那么，abstol 就设置为 1.

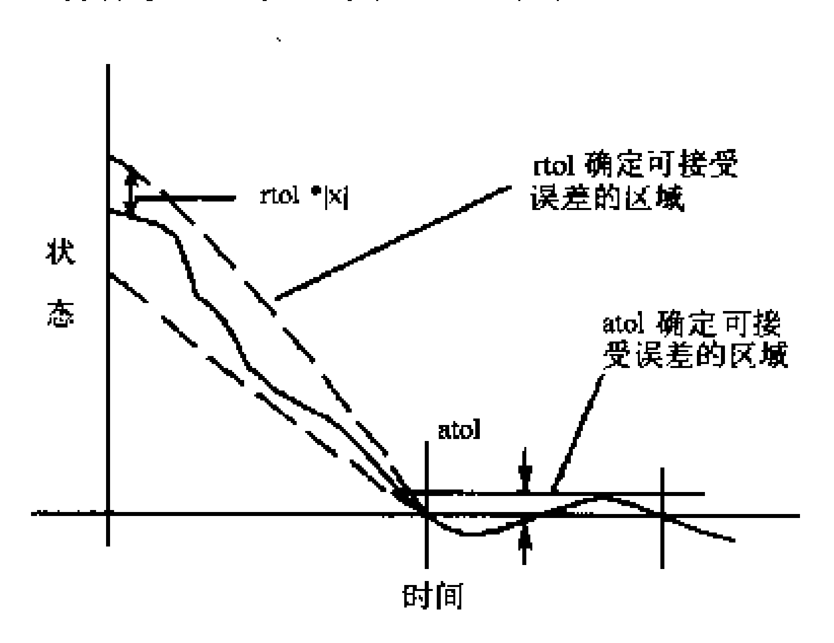


图 4.4 允许误差区域和状态

如果计算设置不合适，可以自己确定适当的设置. 这样可能需要运行好几次仿真，以确定一个合适的绝对误差容限. 如果状态的幅度变化非常大，可能需要给不同的状态指定不同的绝对误差容限. 可以在 Integrator 模块的对话框中做到这一点.

4.2.1.6 ode15s 的最大阶数

ode15s 是基于一到五阶的 NDF 公式的求解器. 尽管公式的阶数越高结果越精确，但稳定性会差一些. 如果模型是刚性的，并且要求有比较好的稳定性，应将最大的阶数减小到 2. 选择 ode15s 求解器时，对话框中会显示这一参数.

可以用 ode23 求解器代替 ode15s，ode23 是定步长、低阶求解器.

4.2.1.7 多任务选项

如果选择定步长求解器，Solver 页面的仿真参数对话框显示一个选项列表，如图 4.5 所示. 可从列表中选择仿真模型.

(1) 多任务(MultiTasking)

如果检测到模块间非法采样率转换，即直接相连模块之间以不同的采样率运算，该模式会出现错误. 在实时多任务系统中，任务间非法采样率转换可能导致当另一个任务需要时，某一任务输出不能用. 通过此类转换检查，多任务模式可以帮助创建现实中的合法的多任务系统模型，模型中何处表示是同时执行的任务.

使用采样率转换(rate transition)模块来减少模型中的非法采样率转换. Simulink 提供了两种这样的模块：Unit Delay 模块和 Zero-Order Hold 模块. 减少非法的慢到快转换，插入一个 Unit Delay 模块，在慢输出端口和快输入端口之间以低速率运行. 减少非法的快到慢转换，插入一个 Zero-Order Hold 模块，在快输出端口和慢输入端口之间以慢速运行.

(2) 单任务(SingleTasking)

该模式不检查模块间的采样率转换. 该模式对于建造单任务系统模型非常有用，在此类系统中，任务同步不是问题.

(3) 自动(Auto)

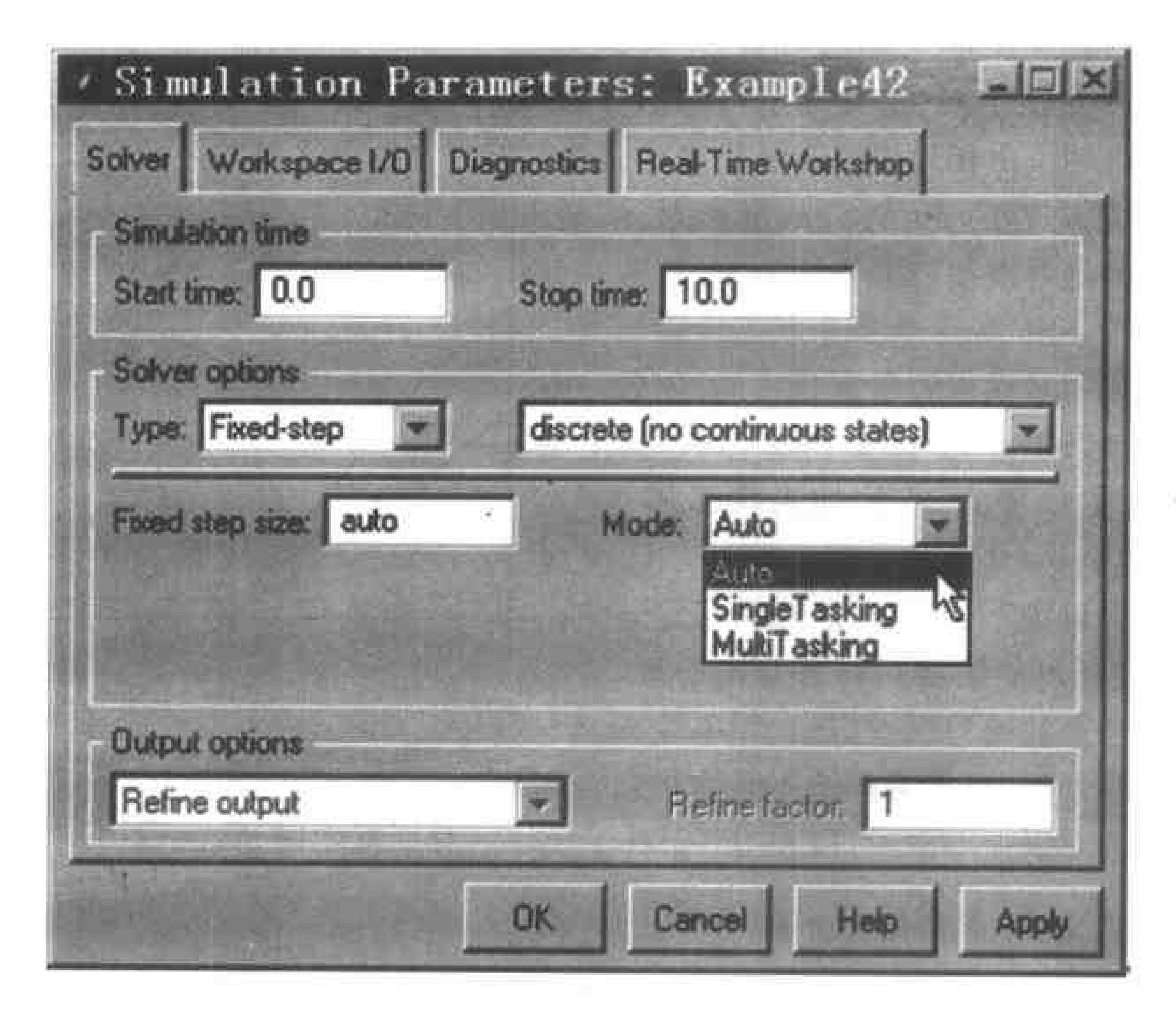


图 4.5　Solver 页面的多任务模式列表

如果模型中所有模块运行于同样的采样率下，Simulink 使用单任务模式；如果模型包含有不同采样率运行的模块，则使用多任务模式.

4.2.1.8　输出选项

可以通过对话框的 Output options 区域控制仿真产生多大的输出. 可以从弹出的 Refine output，Produce additional output，Produce specified output only 三项选项中选择一种.

(1) Refine output(细化输出)

如果仿真输出太粗糙，该选项提供额外的输出点. 该参数提供时间步之间的整数输出点数；如细化因子 2，在时间步输出的同时，在其中间提供输出. 缺省的细化因子是 1.

要获得平滑输出，改变细化因子要比减小步长快得多. 当细化因子改变，求解器通过计算这些点的连续扩展公式，产生额外的点. 改变细化因子，不改变求解器使用的步长.

细化因子用于变步长求解器，而且在使用 ode45 时十分有用. ode45 求解器能够采用大步长，当输出仿真结果图形时，会发现该求解器的输出不够平滑. 如果有此类情况，采用较大的细化因子再重新运行仿真.

(2) Produce additional output(产生额外的输出)

使用该选项，求解器可以在指定的额外的时间产生输出. 选定该选项后，Simulink 就会在 Solver 页面上出现一个输出时间域，在该域上输入一个 MATLAB 表达式来计算额外时间，也可以指定一个额外时间向量. 在额外时间的输出是由连续扩展公式求出的. 与细化因子不同，该选项改变仿真的步长，使得时间步长指定额外输出的时间一致.

(3) Produce specified output only(只产生指定的输出)

该选项只提供指定输出时间的仿真输出,该选项也改变仿真步长,以使时间步长与指定产生输出的时间相符合.这一选项在比较不同的仿真,以保证仿真同时产生输出时非常有用.

(4) 比较输出选项

例 4.3 仍用图 4.3 的模型作例子,步长取 0.5.对上面说到的各选项作一个直观的介绍.

1) 选定步长,在下列时间点输出:

```
tout'
ans =
  Columns 1 through 7
          0    0.2500    0.5000    0.7500    1.0000    1.2500    1.5000
  Columns 8 through 14
     1.7500    2.0000    2.2500    2.5000    2.7500    3.0000    3.2500
  Columns 15 through 21
     3.5000    3.7500    4.0000    4.2500    4.5000    4.7500    5.0000
  Columns 22 through 28
     5.2500    5.5000    5.7500    6.0000    6.2500    6.5000    6.7500
  Columns 29 through 35
     7.0000    7.2500    7.5000    7.7500    8.0000    8.2500    8.5000
  Columns 36 through 41
     8.7500    9.0000    9.2500    9.5000    9.7500   10.0000
```

2)细化因子为 2,将生成下列时间点处的输出:

```
tout'
ans =
  Columns 1 through 7
          0    0.5000    0.7500    1.0000    1.5000    2.0000    2.2500
  Columns 8 through 14
     2.5000    3.0000    3.5000    3.7500    4.0000    4.5000    5.0000
  Columns 15 through 21
     5.2500    5.5000    6.0000    6.5000    6.7500    7.0000    7.5000
  Columns 22 through 28
     8.0000    8.2500    8.5000    9.0000    9.5000    9.7500   10.0000
```

3) 选择 Produce Additional Output,并指定输出时间[0:1:10],将会生成这些时间点上的输出:

```
tout'
ans =
  Columns 1 through 7
          0    0.5000    0.7500    1.0000    1.5000    2.0000    2.2500
```

```
   Columns 8 through 14
     2.5000    3.0000    3.5000    3.7500    4.0000    4.5000    5.0000
   Columns 15 through 21
     5.2500    5.5000    6.0000    6.5000    6.7500    7.0000    7.5000
   Columns 22 through 28
     8.0000    8.2500    8.5000    9.0000    9.5000    9.7500   10.0000
```

4) 选择 Produce Specified Output Only，并指定输出时间[0:0.5:10]，将会生成这些时间点上的输出：

```
tout'
ans =
   Columns 1 through 7
          0    0.5000    1.0000    1.5000    2.0000    2.5000    3.0000
   Columns 8 through 14
     3.5000    4.0000    4.5000    5.0000    5.5000    6.0000    6.5000
   Columns 15 through 21
     7.0000    7.5000    8.0000    8.5000    9.0000    9.5000   10.0000
```

4.2.2 工作空间输入/输出(Workspace I/O)页

可以将仿真的输出保存为工作空间变量，并且可以从工作空间取得输入和初始状态．在仿真参数对话框中，选择 Workspace I/O 标签，这时对话框如图 4.6 所示．

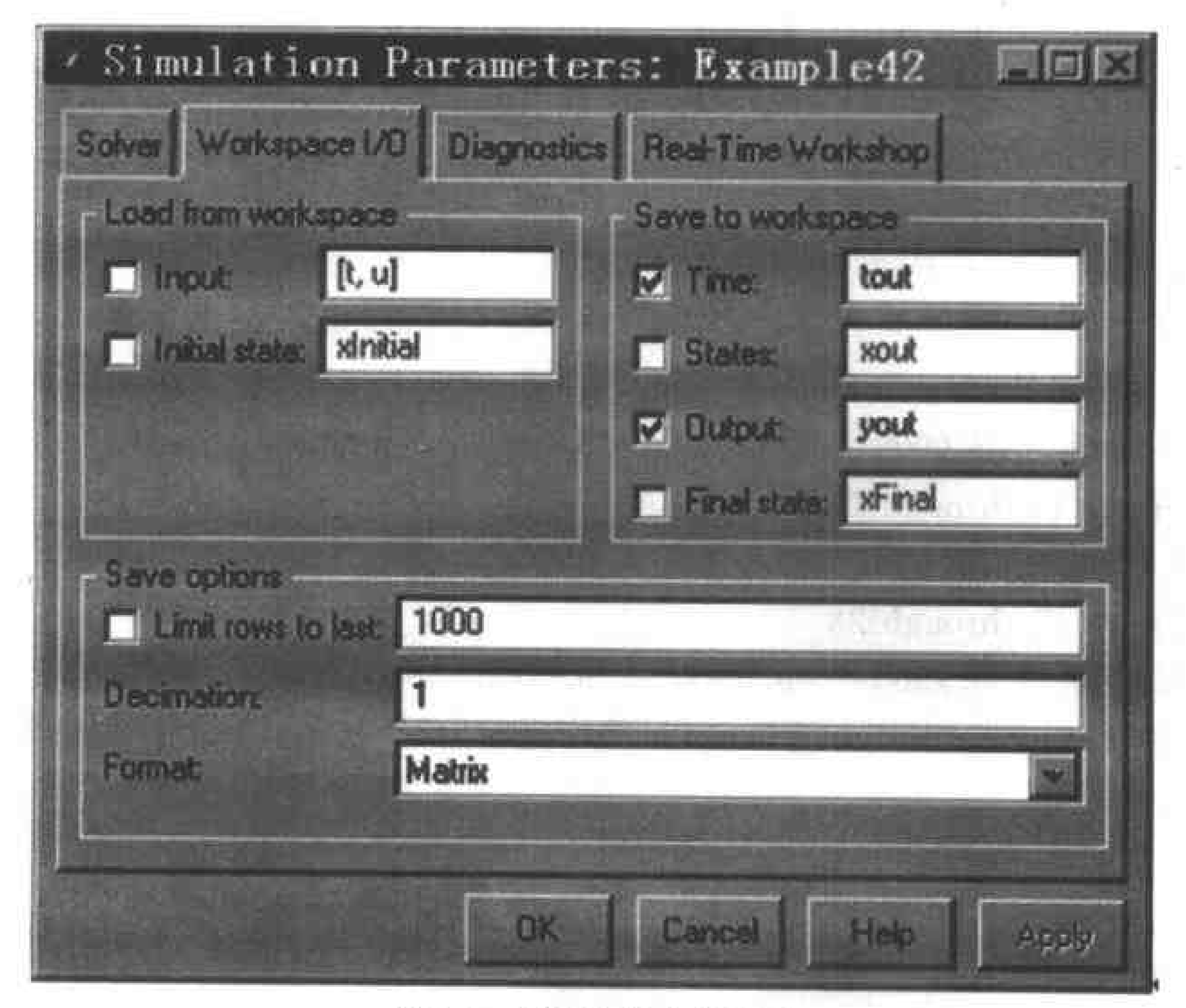


图 4.6　工作空间输入/输出页

4.2.2.1　从基工作空间装入输入

在仿真运行期间,Simulink 可以应用模型基工作空间到模型的顶层输入端口的输入.要指定这一选项,在 Workspace I/O 页面的 Load from workspace 域,选上 Input 核选框,然后,在其后邻近的编辑框中输入外部输入标识(缺省内容为[t, u]),并选择 Apply 按钮.

外部输入可以采用下列任何一种形式.

(1) 外部输入矩阵

外部输入矩阵的第一列必须是升序排列的时间向量,其余列指定输入值.每列代表不同输入模块信号序列,每行则是相应时间的输入值.如果选择了数据插值(interpolate data)选项,必要时 Simulink 对输入值进行线性插值或外推.输入矩阵的总列数必须等于 n+1,其中 n 为进入模型的信号输入端口总数.由于模型缺省的外部输入标识为[t, u],如果在基工作空间中定义了 t 和 u,则不必为模型输入外部输入标识.例如,假设模型有两个输入,其中一个接受两个信号,另一个接受一个信号,并假设工作基工作定义 u 和 t 为

```
t=(0:0.05:1)';
u=[4 * sin(t),cos(t), tan(t)];
```

然后,为模型指定外部输入.

(2) 具有时间的结构(Structure with Time)

Simulink 可以从工作空间中读入结构形式的数据,但其名字必须在 Input 后的文本域中指定.输入结构必须有两个顶层字段:时间和信号.时间字段包含一列仿真时间的向量;信号字段包含子结构数组,每个对应模型的一个输出端口;每个子结构有字段:值;值字段包含相应输入端口的输入列向量.

例 4.4　以图 4.7 所示模型为例,在基工作空间定义向量为

```
ex.time=(0:0.05:10)';
ex.signals(1).values=sin(ex.time);
ex.signals(2).values=2 * cos (ex.time);
ex.signals(2).values=3 *  sin(ex.time). * cos (ex.time);
```

定义并运行上述程序后,将模型指定为外部输入,选择 Input 核选框,并在邻近的文本域中输入结构名:ex,而后应用,并运行仿真,其输出结果如图 4.8 所示.

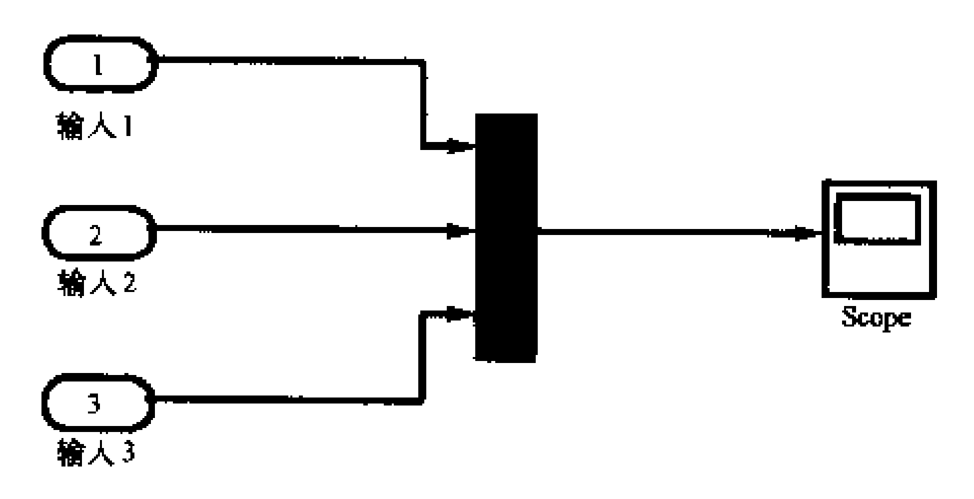


图 4.7　基工作空间结构输入示例模型

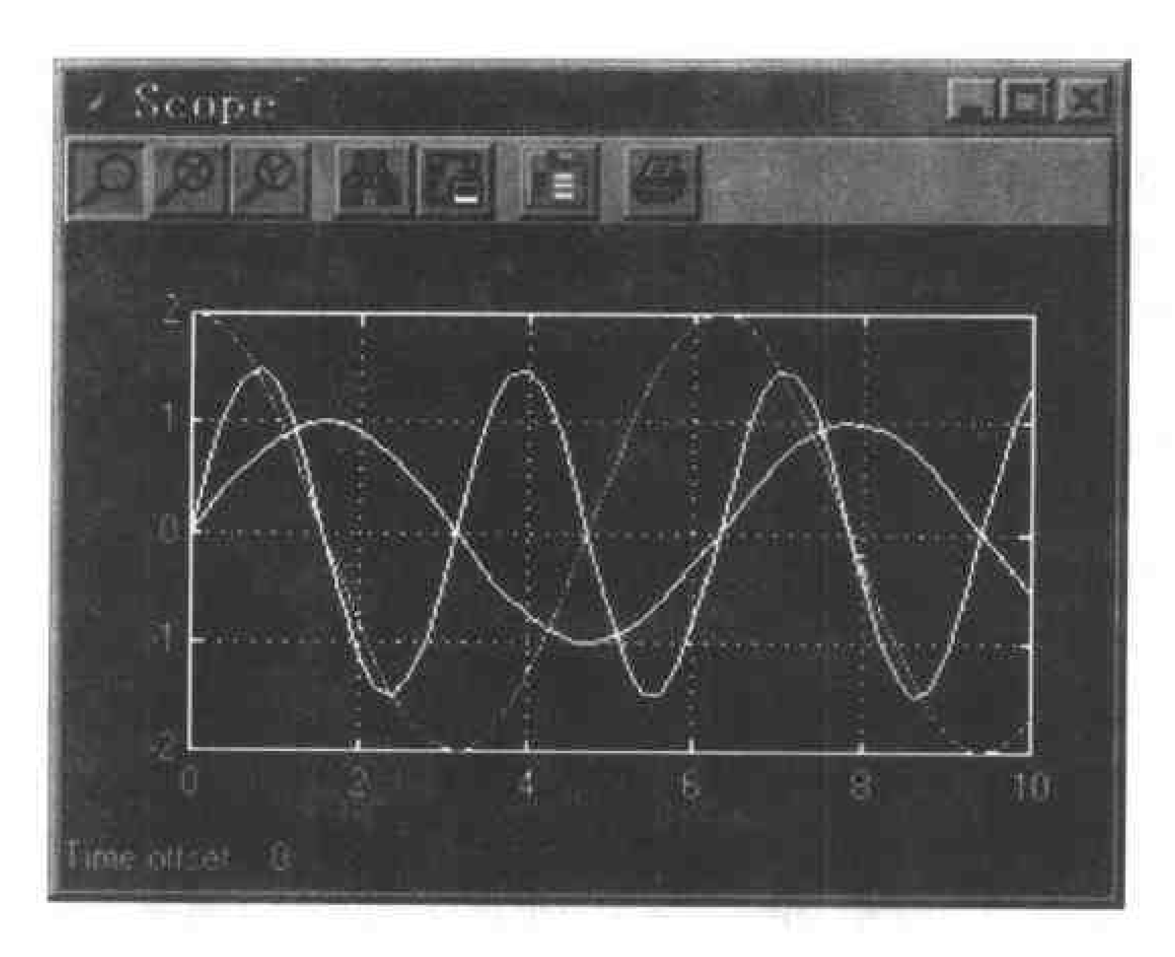


图 4.8　示例输出结果

(3) 结构(Structure)

结构格式与具有时间的结构格式一样,只是其时间字段为空. 如在例 4.4 中,可以指定:ex. time=[]. 此时,Simulink 从输入端口值数组的第一个值读入输入作为第一时间步的输入,第二时间步的值,从第二元素中读入,等等.

(4) 每个端口结构(Per-Port Structures)

这种格式,对于每个端口分别由具有时间的结构或不具有时间的结构组成,每个端口输入数据结构只有一个信号字段. 指定这种选项,在 Input 的文本域中输入用逗号分开的结构名字列表即可.

(5) 外部输入时间表达式(External Input Time Expression)

时间表达式可以是任何 MATLAB 表达式,计算行向量,其长度与模型输入端口的信号数一样. 如,假设一个模型有一个向量输入端口,接受两个信号,而且,假设 timefcn 是一个用户自定义函数,用来返回一两元素长的行向量,下面的输入时间表达式是合法的:

'[3 * cos(t),cos(t)]'

'3 * timefcn(w * t)+7'

Simulink 在仿真的每一步要计算该表达式,将结果值用于模型的输入. Simulink 在运行仿真时,定义了变量 t. 在函数表达式中也可以省略时间变量 t,Simulink 解释 sin 为 sin(t).

4.2.2.2　将结果保存到工作空间

可以通过在该对话框的 Save to workspace 域中选择 Time,States 和/或 Output 核选框指定返回的变量. 指定返回的变量使得 Simulink 将时间、状态和输出值的轨迹输出到工作空间.

要给不同的变量赋值,可在域中各核选框的右边指定不同的变量名. 要将输出写到多

个变量,可指定多个变量名,各变量名之间用逗号分开. Simulink 将仿真时间存放在 Save to Workspace 域中指定名字的向量中.

Simulink 以模型的基采样率保存输出至工作空间. 如果想要以不同的采样率保存输出,应当使用 To Workspace 模块.

可以通过 Save options 域来指定输出存储的格式和限制保存输出的数量.

模型状态和输出的格式选项有:

(1) 矩阵(Matrix)

Simulink 存储模型状态于一个在 Save to Workspace 域中命名的矩阵中,缺省为, xout. 状态矩阵的每一列与一个模型状态相对应,第一行与指定时间的状态对应. 存储模型输出于一个在 Save to Workspace 域中命名的矩阵中,缺省为, yout;每一列与一个模型输出相对应,第一行与指定时间的所有输出对应.

(2) 具有时间的结构(Structure with Time)

Simulink 存储模型输出于一个在 Save to Workspace 域中命名的结构中,缺省为: yout;该结构有两个顶层字段:时间和信号. 时间字段包含仿真时间向量;信号字段包含子结构数组,每个子结构对应一个模型输出端口. 每个子结构包含三个字段:值、标签、模块名. 值字段包含相应输出端口的输出向量;标签字段指定与输出相连的信号标签;模块名字段指定输出端口的名字. Simulink 存储模型的状态于一个结构组成相同的模型输出结构中.

(3) 结构(Structure)

该格式与前面所述的结构基本一样,只是不保存仿真时间于结构的时间字段中.

(4) 每个端口结构(Per-Port Structures)

对于每个输出端口,该格式由单个具有时间的结构和不具有时间的结构组成. 每个输出数据结构有一个信号字段. 指定该选项,在 Output 文本域中以逗号分开输入输出结构名列表.

要限制保存数据的行数,可选择 Limit rows to last 核选框,并指定保存的行数. 要使用抽取(decimation)因子,在 Decimation 文本框中输入数值. 例如,在 Decimation 文本框中输入的值为 2 时,产生的点将每隔一个保存一个.

4.2.2.3 装入和保存状态

系统开始仿真时的初始条件,通常在模块中指定. 也可以在 Workspace I/O 页的 States 域中重新指定,以重载在模块中指定的初始条件.

可以将某次仿真的最终结果保存起来,并可将它们提供给另一次仿真. 如果想保存一个稳定状态的结果并从那个已知的状态重新启动仿真,那么这一特性将非常有用. 状态以该页的 Save options 域中指定的格式保存,其格式包括:

(1) 具有时间的结构

Simulink 保存模型的状态于一个结构中,该结构的名字是由 Save to Workspace 域中的 Final State 字段指定的,如 xFinal. 其结构形式与前面介绍的结构一样,有两个顶层字段:时间和信号. 时间字段包含仿真时间向量;信号字段包含一个子结构数组,每个子结构与具有状态的模块相对应. 每个子结构包含三个字段:值、标签、模块名. 值字段包含相

应模块的状态向量；标签字段可以是连续状态(CState)，或离散状态(Dstate_n)，其中 n 为 1,2,3...，直到相应模块的最大状态数. 模块名字段指定该结构元素表示的模块的名字.

(2) 结构

这种结构与前述的结构一样，只是不保存仿真时间于保存结构中的时间字段.

可以通过选择 Initial State 核选框，并在相邻的域中指定状态向量来装入状态值. 如果该核选框没有被选中或状态向量为空，Simulink 使用模块中指定的初始条件.

可以通过选择 Final State 核选框，并在相邻的编辑框中输入变量名来保存最后的状态(仿真停止时的状态值).

如果想给一个拥有多个状态的模型指定初始条件，必须确定这些状态的次序. 可以通过如下的命令确定一个模型的初始条件和状态的次序：

```
[sizes, x0, xstord]=sys([], [], [], 0)
```

其中：

sys—— 是模型的名字.

sizes——是表示模型特征的一个向量. 只有前两个元素应用于初始条件：sizes(1)是连续状态的数目，sizes(2)是离散状态的数目.

x0——是模块的初始条件.

xstord——是包含模型中拥有状态值的所有模块完整路径名的一个字符串矩阵. xstord 与向量 x0 中模块的次序是相同的.

例 4.5 下面的语句将得到 vdp 模型的初始条件和状态的次序：

```
[sizes, x0, xstord]=vdp([], [], [], 0)
sizes=
     2
     0
x0=
     2
     0
xstord=
     'vdp/Integrator1'
     'vdp/Integrator2'
```

4.2.3 诊断页

可以通过选择 Simulation Parameters 对话框的 Diagnostics 标签来指明在仿真期间遇到一些事件或者条件时希望执行的动作. 对仿真参数对话框的 Diagnostics 页如图 4.9 所示.

对于每一事件类型，可以选择是否需要提示消息，是警告消息还是错误消息. 警告消息不会终止仿真，而错误消息则会中止仿真的运行.

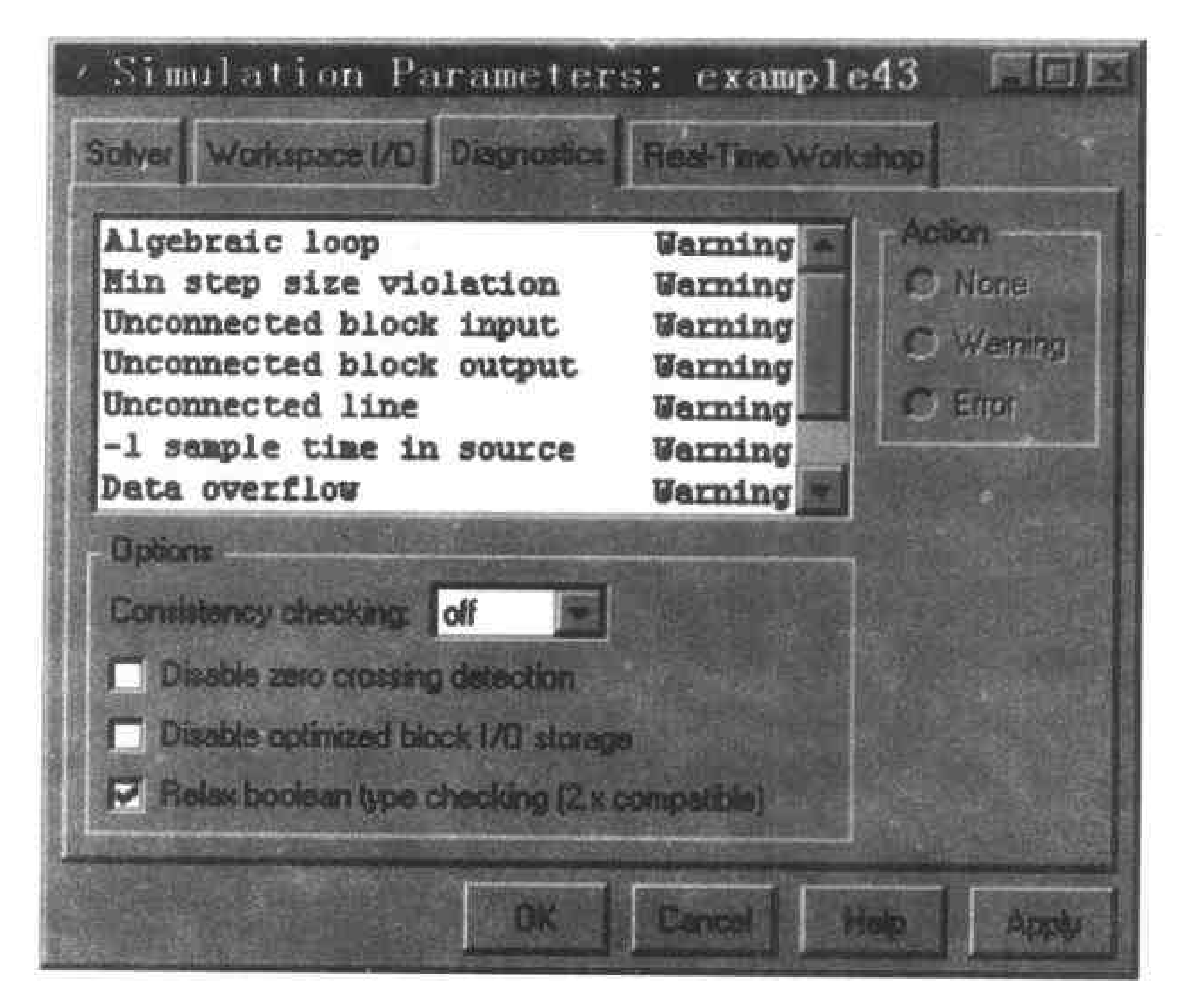


图 4.9 仿真参数对话框的 Diagnostics 页

4.2.3.1 一致性检查

一致性检查是一个调试工具，用它可以验证 Simulink 的 ODE 求解器所做的某些假设. 它的主要用途是确保 S 函数遵循 Simulink 内建模块所遵循的规则. 因为一致性检查会导致性能的大幅度下降(高达 40%)，所以一般应将它设为关的状态. 使用一致性检查可以验证 S 函数，并有助于确定导致意外仿真结果的原因.

为了执行高效的积分运算，Simulink 保存一些时间步的结果，并提供给下一时间步使用. 例如，某一时间步结束的导数通常可以被下一时间步开始时再使用. 求解器利用这一点可以防止多余的导数运算.

一致性检查的另一个目的是保证当模块被以一个给定的 t(时间)值调用时，它产生一常量输出. 这对于刚性求解器(ode23s 和 ode15s)非常重要，因为当计算 Jacobi 行列式时，模块的输出函数可能会被以相同的 t 值调用多次.

如果选择了一致性检查，Simulink 重新计算某些值，并将它们与保存在内存中的值进行比较，如果这些值有不相同的，将会产生一致性错误. Simulink 比较下列量的计算值：

1) 输出；

2) 过零点；

3) 导数；

4) 状态.

4.2.3.2 关闭过零点检测

可以关闭一个仿真的过零点检测. 对于一个有过零点的模型,关闭过零点检测会加快仿真的速度,但是可能影响仿真结果的精度.

这一选项关闭那些本来就有过零点检测的模块的过零点检测. 它不能关闭 Hit Crossing 模块的过零点检测.

4.2.3.3 关闭优化 I/O 存储

选择该选项,将导致 Simulink 为每个模块的 I/O 值分配单独的缓存,而不是重新利用缓存. 这样可以充分增加大模型仿真所需内存的数量. 只有需要调试模型时才选择该选项. 在下列情况下,应当关闭缓存再利用:

1) 调试一个 C-MEX S-函数;

2) 使用浮点 scope 或 display 模块来察看调试模型中的信号.

如果缓存再利用打开,并且试图使用浮点 scope 或 display 模块来显示缓存已被再利用的信号,将会打开一个错误对话框.

4.2.3.4 放松逻辑类型检验

选择该选项,可使要求逻辑类型输入的模块接受双精度类型输入. 这样可保证与 Simulink 3 版本之前的模型的兼容性.

4.3 提高仿真性能和精度

仿值性能和精度由多种因素决定,包括模型的设计和仿真参数的选择.

求解器使用它们的缺省参数值可以使大多数模型的仿真比较精确有效,然而,对于一些模型如果调整求解器和仿真参数将会产生更好的结果. 而且,如果对模型的性能比较熟悉,并且将这些信息提供给求解器,得到的仿真效果将会提高.

4.3.1 加快仿真速度

仿真速度慢的原因有多种. 下面列举其中的一些:

1) 模型中包含有 MATLAB 的 Fcn 模块. 当模型包含有 MATLAB 的 Fcn 模块时,在仿真的每一时间步都会调用 MATLAB 的解释器,这将大大地减慢仿真的速度. 因此应尽可能地使用内建的 Fcn 模块或者 Elementary Math 模块.

2) 模型中包含有 M 文件形式的 S 函数. M 文件形式的 S 函数也将导致在每一时间步调用 MATLAB 的解释器. 可以考虑将 S 函数转换为子系统或者 C-MEX 文件形式的 S 函数.

3) 模型中包含有 Memory 模块. 使用 Memory 模块使得变阶求解器(ode15s 和 ode113)在每一时间步将阶数设为 1 阶.

4) 最大的步长太小. 如果改变了最大步长,可以试试重新使用缺省值(auto)运行仿真.

5）对精度要求太高．缺省的相对容差(0.1%)通常已经足够了．对于状态值趋于零的模型，如果绝对容差设得太小，仿真时状态值在零点附近会花去太多的时间步．

6）时间尺度可能太长．减小时间间隔．

7）问题可能是刚性的，而使用的是非刚性求解器．这时可用 ode15s 试一下．

8）模型使用的采样时间相互之间不成倍数关系．相互之间不成倍数的混合采样时间会导致求解器采用足够小的步长，以保证采样时间符合所有的采样时间要求．

9）模型包含有代数循环．在每一时间步都会反复计算代数循环，因此这会大大地降低仿真的性能．

10）模型中将 Random Number 模块的输出传给了 Integrator 模块．对于连续系统，在 Sources 库中使用 Band-Limited Noise 模块．

4.3.2　改进仿真精度

要检查仿真的精度，仿真运行一段时间以后，减小相对容差到 1e-4 或者减小绝对容差，并重新运行它．比较两次仿真的结果，如果它们之间没有很大的差别，可以确信结果收敛．

如果经过一段时间后，仿真结果变得不稳定，可能是如下原因：

1）系统可能不稳定．

2）如果使用的是 ode15s，可能需要将最大的阶数限制在 2 阶(求解器稳定的最大阶数)，或者试试用 ode23s 求解器．

如果仿真结果看起来不是很精确，可能是：

1）对于一个拥有趋于零的状态值的模型，如果绝对容差设得太大，仿真在零状态值附近花的步数太少．减小绝对容差的大小或者在 Integrator 对话框中为每一个状态分别调整绝对容差的设定．

2）如果减小绝对容差不能有效地提高精度，减少相对容差的大小，减小步长，增加步数．

4.4　通过命令行运行仿真

通过命令行运行仿真与通过菜单运行仿真相比，有如下的一些优点：

1）可以模仿 M 文件和 MEX 文件形式，甚至是 Simulink 模块图形式的模型．

2）可以运行 M 文件来运行仿真，这样，仿真和模块参数可以反复地更改．

在 MATLAB 的命令窗口或 M 文件中，输入仿真命令也可以运行仿真．在进行 Monte Carlo 分析时，可以任意地改变参数，并在一个循环中分别运行仿真，以得到不同参数的结果．可以在命令行中输入 sim 和 set_param 命令来运行仿真．

4.4.1　使用 sim 命令

用该命令运行仿真的完整语法格式为

[t, x, y]=sim(model, timespan, options, ut);

只有 model 参数是必不可少的．命令中没有提供的参数，将使用 Simulation Parame-

ters 对话框进行设置.

参数 options 是一个结构,它提供了附加的仿真参数,包括求解器的名字和误差容限.可以用 simset 命令定义结构参数 options.

4.4.2 使用 set_param 命令

可以使用 set_param 命令来启动、停止、暂停或继续一个仿真,或者更新模块图.同样也可以使用 get_param 命令来检查仿真的状态.set_param 命令的格式为

set_param('sys', 'SimulationCommand', 'cmd')

式中 sys 是系统的名字,cmd 可以是 start,stop,pause,continue 或 update.

get_param 命令的格式为

get_param('sys', 'SimulationStatus')

Simulink 返回'stopped', 'initializing', 'running', 'paused', 'terminating'和'external'等值.

4.4.3 命令 sim

功能:仿真一个动态系统.

语法:[T, X, Y] = sim(Model, Timespan, Options, Ut);

[T, X, Y1, Y2, ..., Yn] = sim(Model, Timespan, Options, Ut);

说明:sim 命令仿真由 Simulink 模型表达的动态系统,对描述模型的普通微分方程系统进行积分.

对于模块图模型,只有 Model 参数是必需的.所有被指定为空矩阵([])的参数值都会采用 Simulation Parameters 对话框来设定仿真参数,而命令中指定的可选参数重置了 Simulation Parameters 对话框中设定的参数.

对于 M 文件和 MEX 文件形式的 S 函数,Model 和 Timespan 参数是必需的.对于连续状态模型,solver 参数是必须被指定的(使用 simset 命令).对于纯离散模型,缺省的 solver 参数值为 VariableStepDiscrete.

命令中参数说明:

T 返回仿真时间向量.

X 返回仿真状态矩阵,包含连续状态和离散状态,连续状态在前,离散状态在后.

Y 返回仿真输出矩阵.对于模块图模型,每一列包含根层 Outport 模块的输出,列号对应着输出端口号.如果某一 Outport 模块的输入是一向量,那么它的输出会占几列而不是一列.

Y1, ..., Yn 只有对于模块图模型使用这种格式.n 是根层 Outport 模块的个数,每一个模块的输出返回在相应的 Yi 中.

Model 模块图的名字.

Timespan 仿真的起始和停止时间.可以指定为下列各种形式之中的一种:

tFinal,指定停止时间,这时起始时间为 0.

[tStart tFinal],指定起始和停止时间.

[tStart OutputTimes tFinal],指定起始时间和停止时间并将时间点返回给T. 通常 T 会包括更多的时间点. OutputTimes 相当于在对话框中选择 Produce additional output.

Options 指定用 simset 命令创建的结构的可选仿真参数.

Ut 可选的模型顶层 Inport 模块的外部输入. Ut 可以是一条或多条 MATLAB 命令表达式或者是一个矩阵. 如果该表达式包含有时间的函数 $u(t)$,MATLAB 在每一时间步都计算它的值. 如果指定的矩阵为 Ut= [T, U1, ..., U2],式中 T= [T1,..., Tn]',第一列必须是递增的时间向量. 剩下的列是相应的输入值. 当需要时,Simulink 在各值之间线性地插补.

例 4.6 下面的命令使用缺省的参数对 vdp 模型进行仿真:

[t, x, y]=sim('vdp')

下一条命令对 vdp 模型进行仿真,仿真时除 Refine 参数外,其它参数使用已经设置好的参数值:

[t, x, y]=sim('vdp', [], simset('Refine', 2));

下一条命令对 vdp 模型仿真 1000 秒钟,保存返回变量的最后 100 行. 仿真只返回 t 和 y,但将最终的状态向量保存在 xFinal 变量中:

[t, x, y] = sim('vdp', 1000, simset('MaxRows', 100, 'OutputVariables', 'ty', 'FinalStateName', 'xFimal'));

参见:simset, simget 命令.

4.4.4 simset

功能:为 sim 命令创建或编辑仿真参数,并且设定求解器的属性.

语法:options =simset(property, value, ...);

options =simset(old_opstruct, property, value, ...);

options =simset(old_opstruct, new_opstruct);

simset

说明:simset 命令创建一个叫作 options 的结构,该结构中指定了有关的仿真参数和求解器属性的值. 结构中没有指定的参数和属性值取它们的缺省值. 要惟一地识别某一参数和属性,只需输入最前面的足够多的字符就可以了,输入的字符不分大小写.

Options =simset(property, value, ...);设置被指名的属性的值,并将它保存在 options 结构中.

Options =simset(old_opstruct, property, value, ...);修改已经存在的结构 old_opstruct 中指名的属性的值.

Options = simset(old_opstruct, new_opstruct);将已经存在的选项结构 old_opstruct 和 new_opstruct 合并成 options,任何 new_opstruct 中定义的属性将重写 old_opstruct 中定义的相同的属性.

不带任何参数的 simset 显示所有属性的名字和它们的值.

下面列举的这些属性和参数是不能够用 get_param 和 set_param 命令来获得或设置它们的值的:

1）AbsTol（绝对误差容限）；正标量，缺省值为 1e-6. 这一标量适用于状态向量的所有元素. AbsTol 只应用于变步长求解器.

2）Decimation（输出变量的降采样因子）；正整数，缺省值为 1. 降采样因子适用于返回变量 t，x 和 y. 值为 1 的降采样因子将返回所有时间点，值为 2 的降采样因子每隔一个返回一个时间点.

3）DstWorkspace（赋变量于何处）；'base'或{'current'}或'parent'. 这一属性确定在 To Workspace 模块中定义为返回变量或输出变量的任何变量赋的值保存于哪一个工作空间中.

4）FinalStateName（最终状态变量的名字）；字符串，缺省为''. 这一属性指定 Simulink 在仿真结束时保存模型状态的变量的名字.

5）FixedStep（定步长）；正标量. 这一属性只适用于定步长求解器. 如果模型中包含有离散的组件，缺省值将是基采样时间；否则将是仿真间隔的五十分之一.

6）InitialState（连续或离散状态的初始值）；向量，缺省为[]. 初始状态包含连续状态（如果有的话）和离散状态（如果有的话），连续状态在前，离散状态在后. InitialState 取代模型中指定的初始状态. 缺省的空矩阵使得初始状态值取模型中指定的值.

7）InitialStep（建议的初始步长）；正标量，缺省为'auto'. 这一属性只适用于变步长求解器. 缺省情况下，求解器自动地确定初始步长.

8）MaxOrder（Ode15s 的最大阶数）；可以是 1、2、3、4 或 5，缺省为 5. 这一属性只适用于 ode15s.

9）MaxRows（输出行数的限制）；非负整数，缺省为 0. 这一属性限制 t，x 和 y 中返回的行数为最后的 MaxRows 个时间点. 如果使用缺省的 0 值，将没有限制.

10）MaxStep（步长的最大值）；正标量，缺省为'auto'. 这一属性只适用于变步长求解器，并且缺省的值是仿真间隔的五十分之一.

11）OutputPoints（确定输出点数）；'specified'或者'all'，缺省的是'specified'. 当设为'specified'时，求解器只生成在 timespan 中指定的时间点的输出 t，x 和 y. 当设为 all，t，x 和 y 还包含求解器所有采用的时间步.

12）OutputVariables（设置输出变量）；可以是 txy，tx，ty，xy，t，x 或 y，缺省为 txy. 如果在属性字符串中没有"t"，"x"或"y"，求解器将在相应的输出 t，x 或 y 中生成一个空的矩阵.

13）Refine（输出细化因子）；正整数，缺省为 1. 这一属性通过指定的因子增加输出点的数目，以产生更为光滑的输出. Refine 只适用于变步长求解器. 如果指定了输出的时间，它将被忽略.

14）RelTol（相对误差容限）；正标量，缺省为 1e-3. 这一属性适用于状态向量的所有元素. 每一积分步的估计误差满足：e(i)<=max(RelTol * abs(x(i))，AbsTol(i))，这一属性只适用于变步长求解器.

15）Solver（时间增加方法）；可以是 VariableStepDiscrete，ode45，ode23，ode113，ode15s，ode23s，FixedStepDiscrete，ode5，ode4，ode3，ode2 或 ode1. 这一属性指定在前面的时间中使用的求解器.

16）SrcWorkspace（在何处计算表达式）；可以是'base'，'current'或'parent'，缺省为

'base'. 这一属性指定在哪一个工作空间中计算模型中定义的 MATLAB 表达式.

17) Trace(跟踪工具)：可以是'minstep','siminfo'或'compile',缺省为空''. 这一属性启用仿真跟踪工具(可以取一或多个,多个时用逗号分开).

指定'minstep'跟踪标志时,当结果改变太快以致于变步长求解器不能确定步长和满足误差容限时,仿真将会结束. 缺省情况下,Simulink 给出一条警告信息并继续运行仿真.

指定'siminfo'标志,使得在仿真开始时提供仿真参数的一个简短的总结.

指定'compile'标志,显示一个模块图模型的编译信息.

18) ZeroCross(开/关过零点定位)：'on'或'off',缺省为'on'. 这一属性只适用于变步长求解器. 如果设为'off',变步长求解器将不会检测过零点,即使模块本来就有过零点检测功能. 求解器调整它们的步长只是为了满足误差容限.

例 4.7　下面的命令生成一个名为 myopts 的选项结构,它定义了 MaxRows 和 Refine 参数的值,其它的参数使用缺省值：

```
myopts=simset('MaxRows',100,'Refine',2);
```

下面的命令使用 myopts 中定义的参数对 vdp 模型仿真 10 秒钟：

```
[t,x,y]=sim('vdp',10,myopts);
```

单独使用 simset 命令：

```
simset
          Solver: ['VariableStepDiscrete' |
                   'ode45' | 'ode23' | 'ode113' | 'ode15s' | 'ode23s' |
                   'FixedStepDiscrete' |
                   'ode5' | 'ode4' | 'ode3' | 'ode2' | 'ode1' ]
          RelTol: [ positive scalar {1e-3} ]
          AbsTol: [ positive scalar {1e-6} ]
          Refine: [ positive integer {1} ]
         MaxStep: [ positive scalar {auto} ]
     InitialStep: [ positive scalar {auto} ]
        MaxOrder: [ 1 | 2 | 3 | 4 | {5} ]
       FixedStep: [ positive scalar ]
    OutputPoints: [ {'specified'} | 'all' ]
 OutputVariables: [ {'txy'} | 'tx' | 'ty' | 'xy' | 't' | 'x' | 'y' ]
      SaveFormat: [ {'Matrix'} | 'Structure' | 'StructureWithTime']
         MaxRows: [ non-negative integer {0} ]
      Decimation: [ positive integer {1} ]
    InitialState: [ vector {[]} ]
  FinalStateName: [ string {''} ]
           Trace: [ comma separated list of 'minstep', 'siminfo', 'compile' {''}]
    SrcWorkspace: [ 'base' | {'current'} | 'parent' ]
    DstWorkspace: [ 'base' | {'current'} | 'parent' ]
```

ZeroCross：[{'on'} | 'off']

参见：sim，simget 命令

4.4.5 simget

功能：取得选项结构的属性和参数.

语法：Struct =simget(model)

Value =simget(model, property)

说明：simget 命令获取指定的 Simulink 模型中的仿真参数和求解器的属性值. 如果参数和属性被用一个变量名定义过，simget 返回变量的值，而不是它的名字；如果工作空间中不存在该变量，Simulink 将发布一条错误消息.

Struct=simget(model)；返回指定模型的当前选项结构. 选项结构是用 sim 和 simset 命令定义的.

Value=simget(model, property)；从模型中提取被指名的仿真参数和求解器属性的值.

Value=simget(OptionStructure, property)；从 OptionStructure 中提取被指名的仿真参数和求解器属性的值，如果结构中没有指定值将返回一个空的矩阵. Property 可以是一个包含你感兴趣的参数和属性列的单元数组. 如果用的是单元数组，输出同样是单元数组.

要惟一地识别某一参数和属性，只需输入最前面的足够多的字符就行了，输入的字符不分大小写.

例 4.8　下面的命令取得 example43 模型的选项结构：

```
op=simget('example43')
op =
           AbsTol: 'auto'
            Debug: 'off'
       Decimation: 1
     DstWorkspace: 'current'
   FinalStateName: ''
        FixedStep: 'auto'
     InitialState: []
      InitialStep: 'auto'
      MaxOrder:5
       SaveFormat: 'StructureWithTime'
          MaxRows: 0
          MaxStep: 'auto'
     OutputPoints: 'all'
  OutputVariables: 'ty'
           Refine: 1
           RelTol: 0.0010
```

```
          Solver: 'ode45'
    SrcWorkspace: 'base'
           Trace: ''
       ZeroCross: 'on'
```

下面的命令取得 vdp 模型的 Refine 属性的值：

```
refine=simget('vdp','Refine');
refine =
     1
```

参见：sim，simset 命令

第五章　仿真结果分析

仿真结果分析是进行建模与仿真的一个重要环节.结果分析有助于模型的改进、完善,同时结果分析也是仿真的主要目的.Simulink 也提供了一些仿真结构分析的函数和命令.读者可以根据自己模型的特点和需要,利用各种工具箱来构造自己的仿真分析程序.下面介绍常用的 Simulink 仿真结果的分析方法.

5.1　观察输出轨迹

绘制从 Simulink 输出的轨迹的方法有:将信号输入 Scope 或 XY Graph 模块;将输出写入变量并用 MATLAB 绘图命令;使用 To Workspace 模块将输出写入工作空间并用 MATLAB 绘图命令绘出结果.

5.1.1　使用 Scope 模块

可以在仿真期间在 Scope 模块中显示输出轨迹.如图 5.1 所示的这一简单模型说明了如何使用 Scope 模块.图 5.2 是两个 Scope 模块的显示结果.

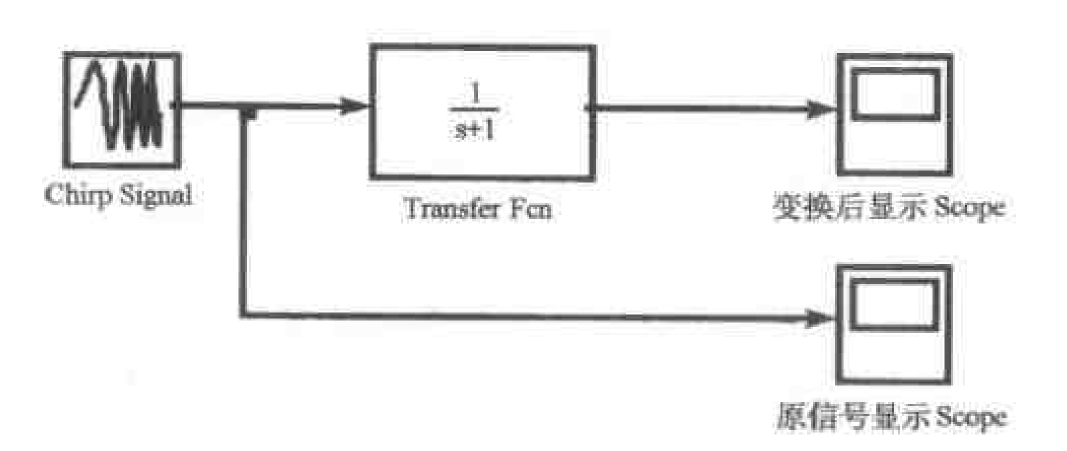


图 5.1　使用 Scope 模块显示轨迹

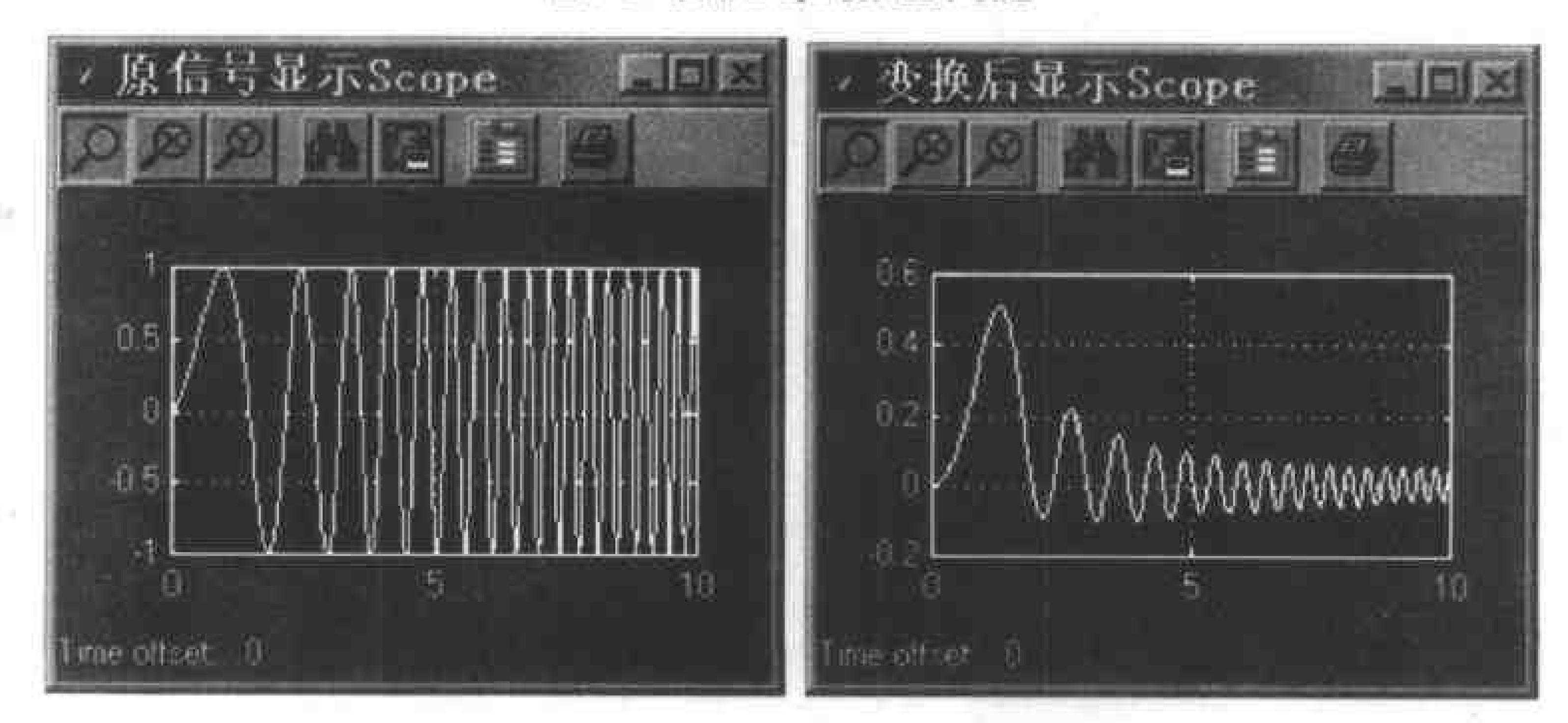


图 5.2　Scope 显示轨迹窗口

Scope 窗口中显示输出的轨迹. 使用 Scope 模块可以对感兴趣的部分进行放大,也可以将数据存入工作空间.

使用 XY Graph 模块可以绘出对比图.

5.1.2　使用返回变量

通过返回时间和输出历史数据,可以用 MATLAB 的绘图命令显示并标注输出轨迹.

图 5.3 所示模型中标为 Out1 的模块是 Signals & Systems 库中的一个输出端口模块. 输出轨迹 yout 是通过积分计算器返回的.

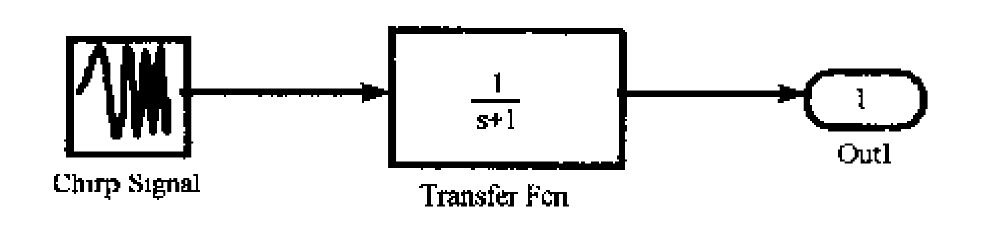


图 5.3　示例模型

可以通过在 Simulation Parameters 对话框的 Workspace I/O 页面中指定变量时间(tout)、输出(yout)和状态(xout)后,再通过 Simulation 菜单运行仿真. 然后可以在 MATLAB 命令行中使用下面的命令,绘出结果.

```
plot(tout,yout)
```

图 5.4 是上述命令的运行结果图.

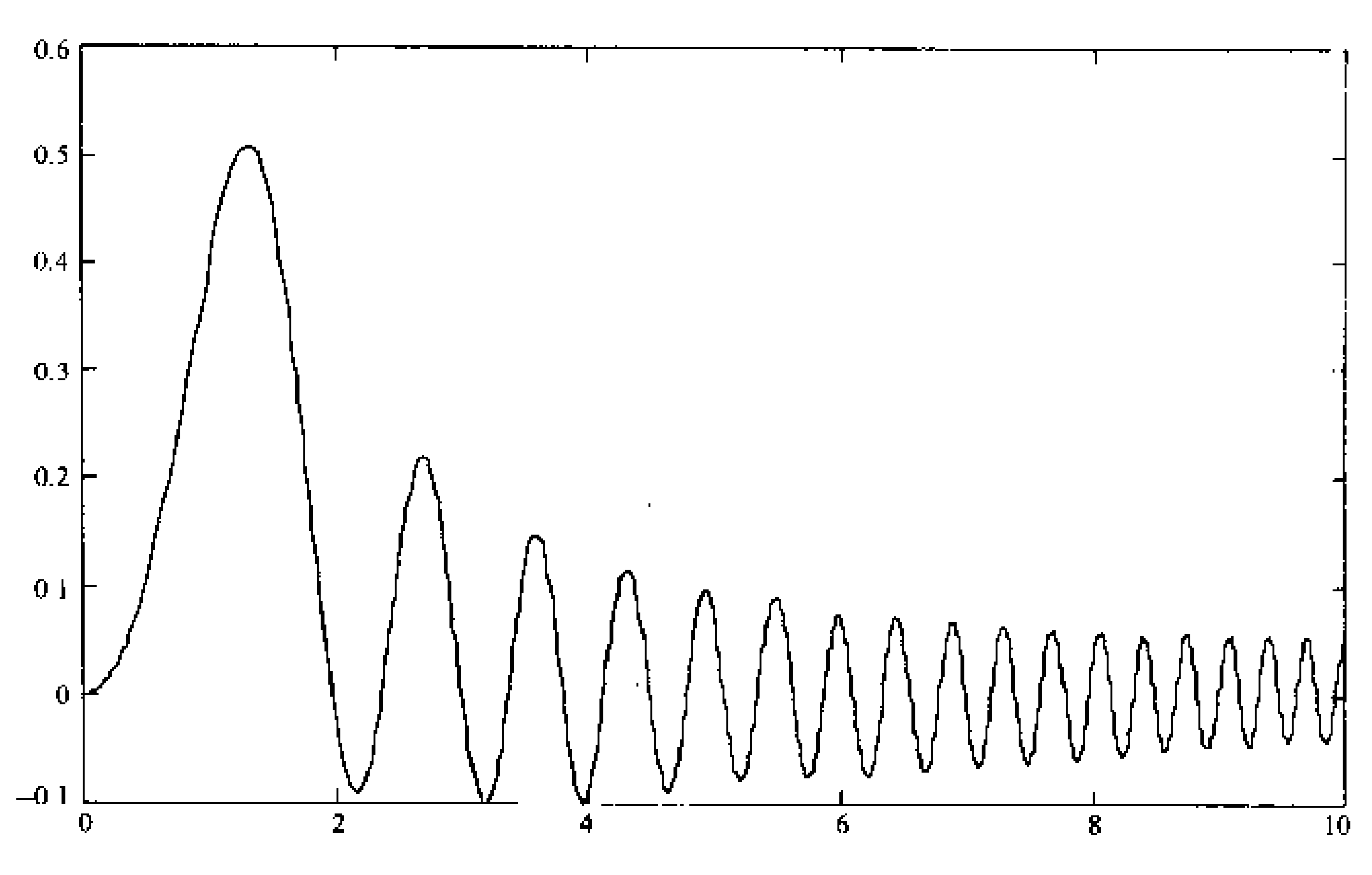


图 5.4　用 MATLAB 绘图命令对返回变量画图

5.1.3　使用 To Workspace 模块

使用 To Workspace 模块可以返回输出轨迹到 MATLAB 的工作空间. 如图 5.5 所示

的模型说明了这一用法.

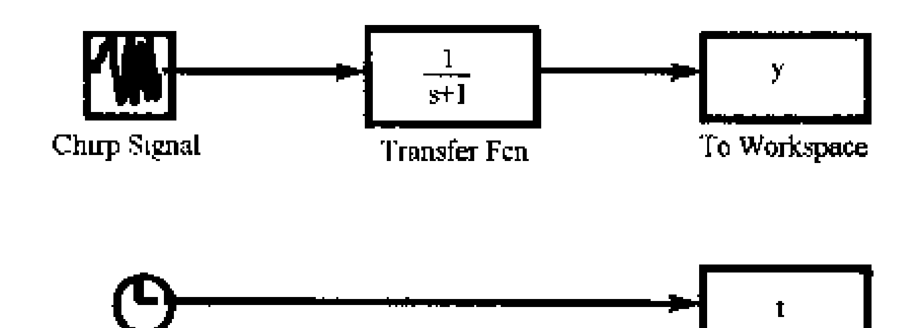


图 5.5 使用 To Workspace 模块将结果输出到工作空间

变量名 y 和 t 在模块参数对话框中设置. 当仿真结束时将在工作空间中显示变量 y 和 t. 通过将 Clock 模块输入 To Workspace 模块保存时间向量. 对于菜单驱动的仿真,通过在 Simulation Parameters 对话框的 Workspace I/O 页面中输入时间的变量名可以获得时间向量,或通过 sim 命令返回它.

To Workspace 模块能够接受向量的输入,在返回的工作空间变量中,每一输入元素的轨迹存于一个列向量中.

5.2 线性化

Simulink 提供 linmod 和 dlinmod 函数以提取线性模型,模型的形式存为状态空间矩阵 A,B,C 和 D. 状态空间矩阵用下面的式子描述输入输出之间的关系:

$$\begin{aligned} x' &= \mathrm{A}x + \mathrm{B}u \\ y &= \mathrm{C}x + \mathrm{D}u \end{aligned} \tag{5.1}$$

式中 x、u 和 y 分别是状态、输入和输出向量. 图 5.6 所示模型名为 linearmod:

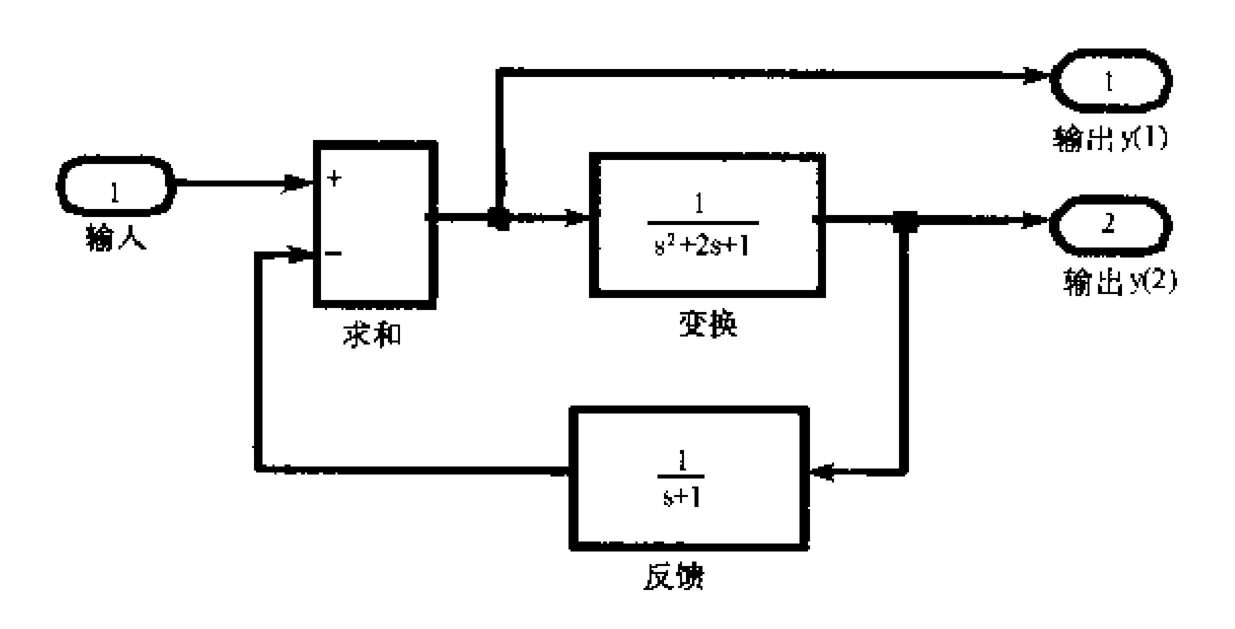


图 5.6 模型 linearmod

例 5.1 要提取图 5.6 所示 Simulink 系统的线性模型,输入命令:

```
[A,B,C,D]=linmod('linearmod')
A =
    -1     0     1
    -1    -2    -1
     0     1     0
```

```
B =
        0
        1
        0
C =
       -1   0   0
        0   0   1
D =
        1
        0
```

必须用 Signals & Systems 库中的 Inport 和 Outport 模块定义输入和输出，Source 和 Sink 模块不作为输入和输出. Inport 模块可以用来使用 Sum 模块连接 Source 模块.

一旦数据成了状态空间形式或者转变成了 LTI 对象，就可以使用 Control System Toolbox 的函数进行进一步的分析.

(1) 转换成 LTI 对象

sys=ss(A,B,C,D)

例 5.2　对于图 5.6 所示模型，使用该命令后，结果为

```
sys=ss(A,B,C,D)
a =
            x1    x2    x3
      x1    -1     0     1
      x2    -1    -2    -1
      x3     0     1     0
b =
            u1
      x1     0
      x2     1
      x3     0
c =
            x1    x2    x3
      y1    -1     0     0
      y2     0     0     1
d =
            u1
      y1     1
      y2     0
```

(2) Bode 相位、幅度与频率图

bode(A, B, C, D)或者 bode(sys)

例 5.3　对例 5.1 或 5.2 结果使用 bode(sys)命令，得图 5.7.

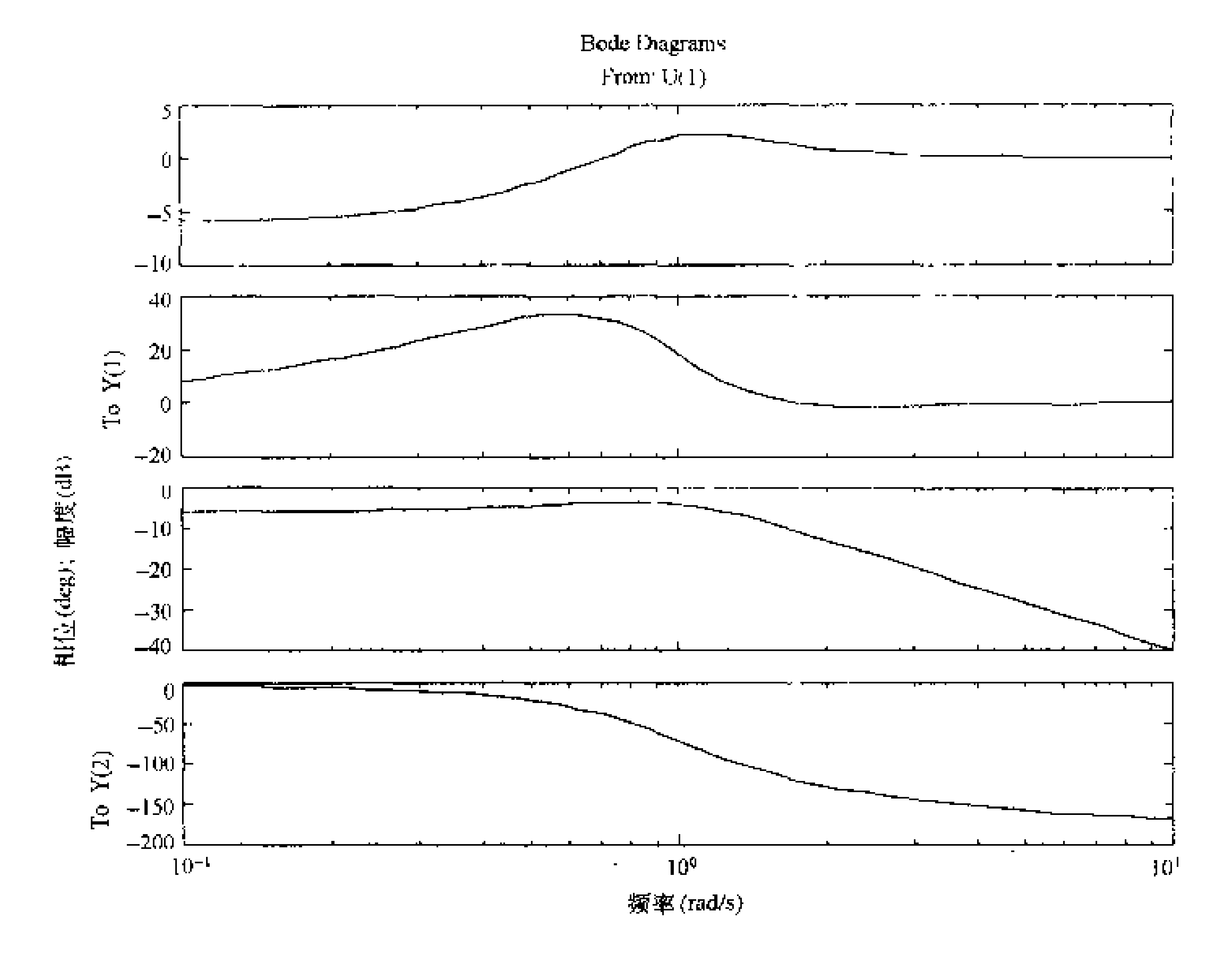


图 5.7 Bode 相位、幅度与频率图

(3) 线性化的时间响应

step(A, B, C, D)或者 step(sys)

impulse(A, B, C,D)或者 impulse(sys)

lsim(A, B, C, D, u, t)或者 lsim(sys, u, t)

例 5.4 对例 5.1 或 5.2 的结果，使用 step(A, B, C, D)命令，得到图 5.8；使用 impulse(A, B, C,D),得到图 5.9.

Control System Toolbox(控制系统工具箱)和 Robust Control Toolbox(鲁棒控制工具箱)中其它的一些函数可以用作线性控制系统设计.

如果模型是非线性的，可能要选择一个运算点，以确定在何处提取线性模型. 非线性模型对提取点的扰动大小比较敏感. 这些都必须进行选择，以在截断误差和舍入误差之间取得折衷平衡. linmod 函数的其它参数是指定运算点和扰动点的.

[A, B, C, D]=linmod('systemname', x, u, pert, xpert, upert)

对于离散系统或连续与离散混合的系统，使用 dlinmod 函数进行线性化. 调用 dlinmod 的语法与调用 linmod 的语法相同，但右边第二个参数必须包含采样时间，以确定在什么时候执行线性化.

使用 linmod 线性化一个包含有 Derivative 或者 Transport Delay 模块的模型，将会遇到麻烦. 在线性化之前，要用特别设计的模块来代替这些模块以避免出现问题. 这些模块在 Simulink Extras 库的 Linearization 子库中. 可以通过打开 Blocksets & Toolboxes 图标来访问 Extras 库.

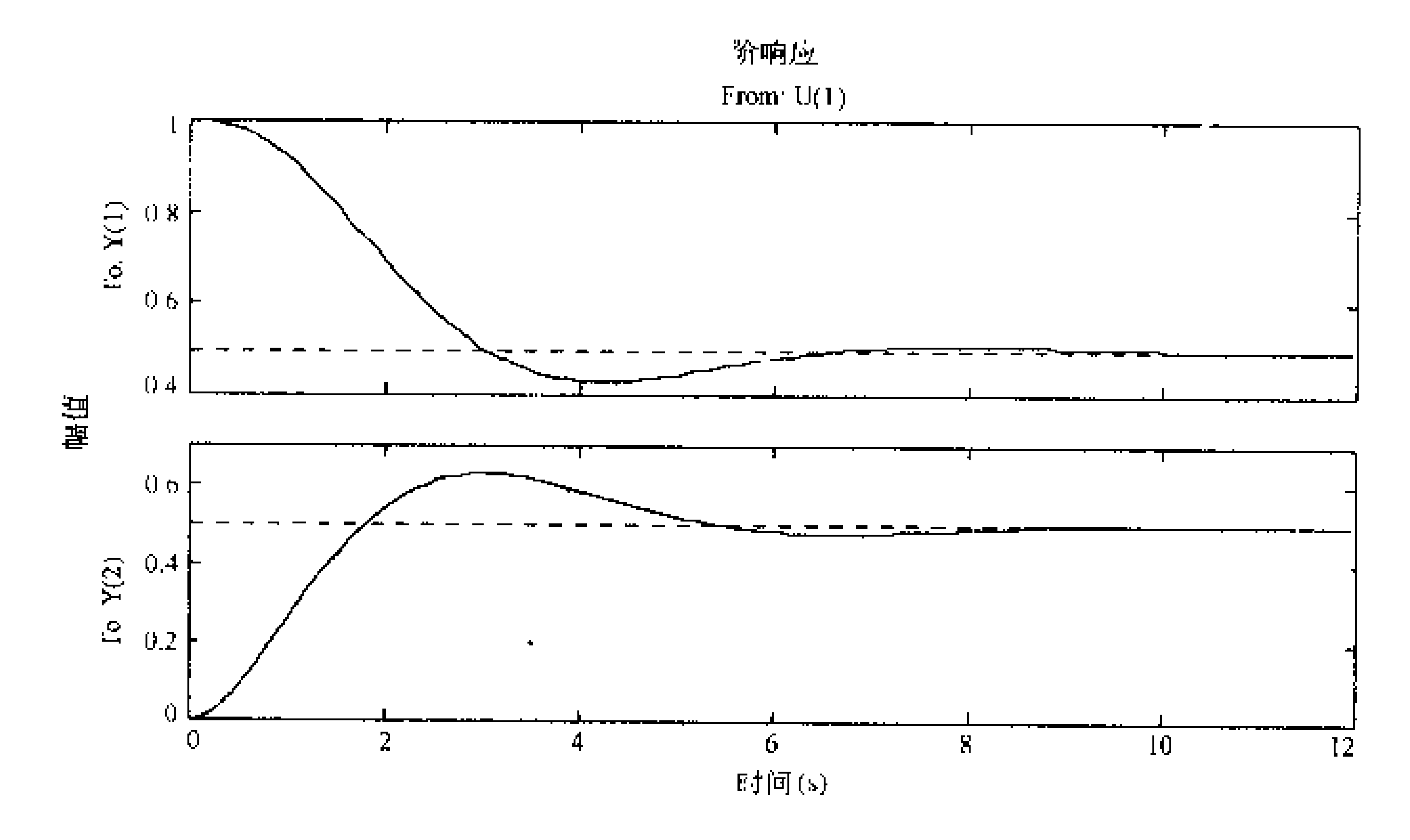


图 5.8　线性化阶响应

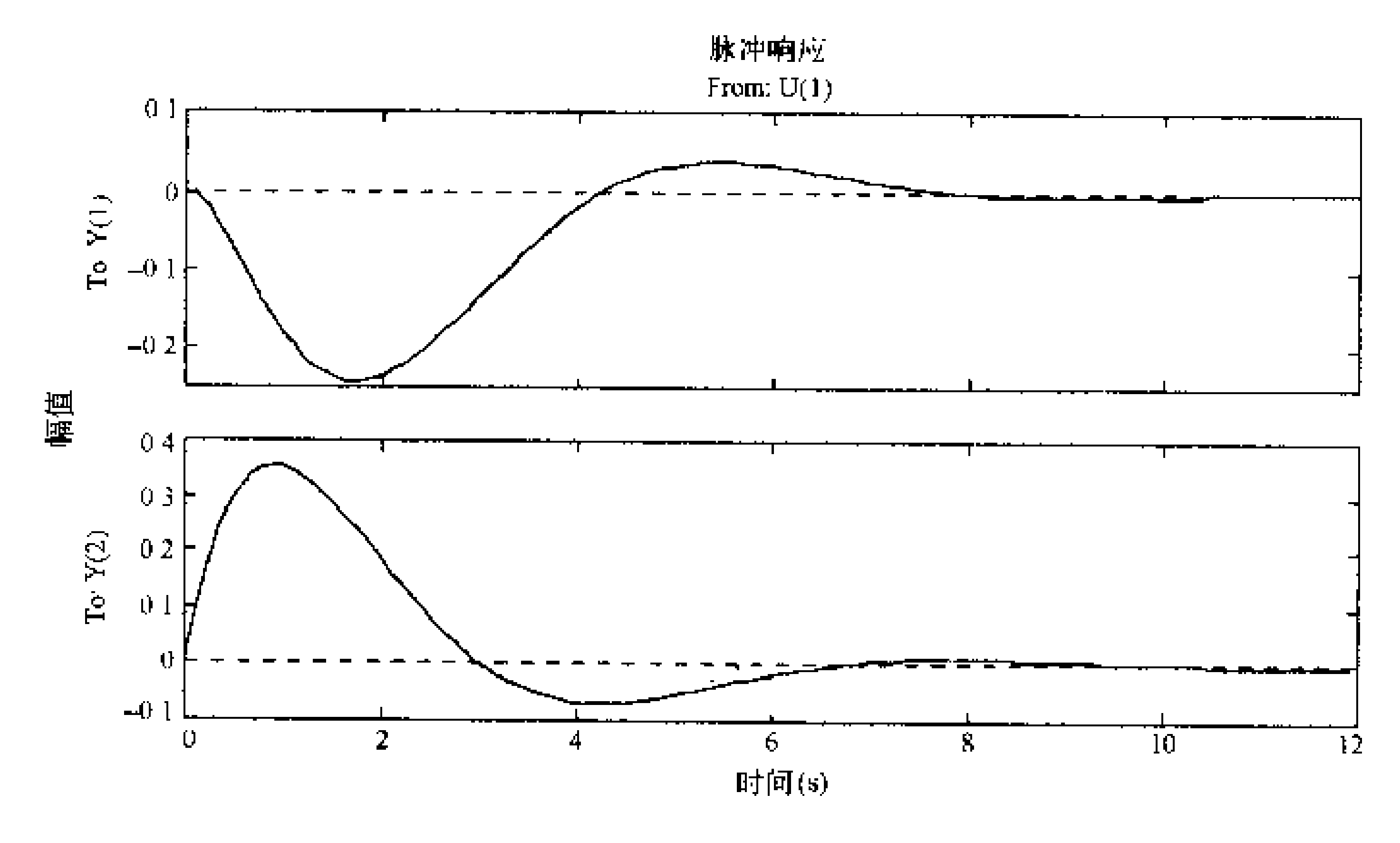


图 5.9　线性化脉冲响应

对于 Derivative 模块，使用 Switched 导数来进行线性化.

对于 Transport Delay 模块，使用 Switched 传输延迟以线性化，使用这一模块要用到控制系统工具箱.

当使用 Derivative 模块时，也可以将导数项与其它模块合并. 如果有一个 Derivative 模块与一个 Transfer Fcn 模块串联，用一个如下形式的单一的 Transfer Fcn 模块实现将会更好一些：

$$\frac{s}{s+a} \tag{5.2}$$

在图 5.10 中，图左边的两个模块可以用图右边的一个模块代替：

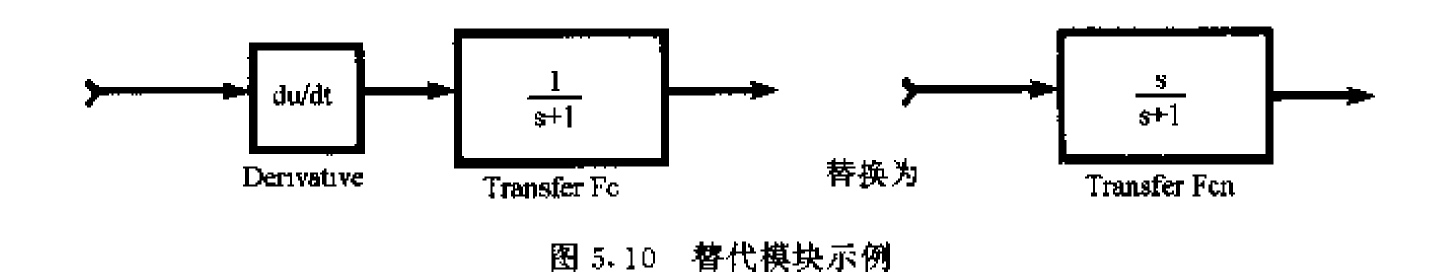


图 5.10　替代模块示例

5.3　平衡点的确定(trim)

Simulink 中的 trim 函数用来确定稳态平衡点.

例 5.5　考虑如图 5.6 所示的名为 linearmod 的模型. 使用 trim 函数找到使两个输出都为 1 的输入值和状态值. 首先对状态变量 x 和输入值 u 进行初始估计,然后设定好输出 y 的希望值:

```
x=[0; 0; 0];
u=0;
y=[1; 1];
```

下面的三条语句使用索引变量指明哪些变量是不变的,哪些变量是可变的:

```
ix=[];        %任何状态值可变
iu=[];        %任何输入可变
iy=[1; 2];    %两个输出不能变
```

调用 trim 返回结果. 由于圆整的原因不同的计算机计算的结果可能有些不同.

```
[x, u, y, dx]=trim('linearmod',x, u, y, ix, iu, iy)
y =
    1
    1
x =
    1.0000
         0
    1.0000
u =
    2.0000
y =
    1.0000
    1.0000
dx =
  1.0e-015 *
  -0.2220
   0.2220
        0
```

对于平衡点问题可能没有结果,如果是这样的话,trim 首先试着将导数设为 0,以最

小化最大偏差,并将它作为结果返回.

5.4 线性化分析函数(linfun)

功能:在某一运算点附近提取系统的线性状态空间模型.

语法:[A, B, C, D]=linfun('systemname')

[A, B, C, D]=linfun('systemname', x, u)

[A, B, C, D]=linfun('systemname', x, u, pert)

[A, B, C, D]=linfun('systemname', x, u, pert, xpert, upert)

说明:

表 5.1 给出了各个参数说明.

表 5.1 线性化分析函数的参数说明

参 数	说 明
linfun	linmod,dlinmod 或 linmod2
sys	将要提取线性化模型的 Simulink 系统的名字
x, u	状态和输入向量.如果被指定,它们设定提取线性模型的运算点
pert	x 和 u 要用到的可选的标量扰动因子.如果没有指定,缺省值为 1e-5
xpert, upert	可选的向量用来分别设定每一个状态和输入的扰动程度.如果指定了该参数,将忽略参数 pert.第 i 个状态 x 扰动至 x(i)+xpert(i);第 j 个输入 u 扰动至 u(j)+upert(j)

函数 linmod 从表达为 Simulink 模型的普通微分方程的系统中提取线性模型.函数 linmod 以状态空间 A,B,C,D 的形式返回线性模型,它们描述了输入与输出的线性化关系,如式(5.1).

在 Simulink 的模块图中,使用 Inport 和 Outport 模块来指明输入和输出.

[A, B, C, D]=linmod('systemname');获得 sys 在状态变量 x 和输入 u 为 0 时的运算点附近的线性模型.

函数 linmod 在运算点附近扰动状态值,以确定状态导数和输出(Jacobi 行列式)的变化速度.这一结果将用来计算状态矩阵.每一状态 $x(i)$ 扰动成

$$x(i) + \Delta(i) \tag{5.3}$$

式中 $\Delta(i)=\delta(1+|x(i)|)$.

同样第 j 个输入扰动成

$$u(j) + \Delta(j) \tag{5.4}$$

式中 $\Delta(j)=\delta(1+|u(j)|)$.

5.4.1 离散时间系统的线性化

函数 dlinmod 可以在任何给定的采样时间处线性化离散、多采样率和连续与离散混合的系统.调用 dlinmod 与调用 linmod 的语法相同,只不过需要插入采样时间作为第二

个参数，以确定在哪些时间点执行线性化. 例如，命令：

[Ad，Bd，Cd，Dd]=dlinmod('systemname'，Ts，x，u)；

在采样时间 Ts 和由状态向量 x 和输入向量 u 决定的运算点处，产生一个离散的状态空间模型. 要得到一个离散系统的连续模型的近似，将 Ts 设为 0.

对于由线性、多采样率和连续模块组成的系统，假如：

1)Ts 是系统中所有采样时间的整数倍；

2)Ts 不小于系统中最慢的采样时间；

3)系统是稳定的.

函数 dlinmod 在转换后的采样时间 Ts 处，产生有着相同频率和时间响应（对于常量输入）的线性模型.

当这些条件不满足时，也有可能得到有效的线性模型.

计算线性化的矩阵 Ad 的特征值，可以知道系统的稳定性. 如果 Ts>0，并且特征值位于单位圆内则系统是稳定的，即

all(abs(eig(Ad)))<1

同样，如果 Ts＝0，并且特征值在左半平面内系统也是稳定的，即

all(real(eig(Ad)))<0

当系统不稳定，而且采样时间不是其它采样时间的整数倍时，dlinmod 生成 Ad 和 Bd 矩阵，它们可能是复数矩阵. 在这种情况下，从 Ad 矩阵的特征值仍然可以知道其稳定性.

使用 dlinmod 命令可以将系统的采样时间转换为其它的值，或者将线性离散系统转换为连续系统，反之亦然. 使用 bode 函数可以了解连续或离散系统的频率响应.

5.4.2 线性化的高级形式

程序 linmod2 提供了线性化的一种高级形式. 这一程序花的时间比 linmod 长，但可以产生更为精确的结果.

调用 linmod2 的语法与调用 linmod 的语法相似，但功能不一样. 例如，linmod2('systemname'，x，u)同 linmod 一样生成线性模型；然而，它的每一个状态空间矩阵元素的扰动水平是被分别设定的，以将圆整和截断误差减到最小.

函数 linmod2 试图平衡圆整误差（产生于小的扰动水平，它产生的误差与有限精度数学有关）和截断误差（产生于大的扰动水平，它使得分段线性近似失效）.

[A，B，C ，D]＝linmod2('systemname'，x，u，pert)，变量 pert 指明了可使用的最低水平的扰动，缺省值为 1e-8. linmod2 有一个优点，就是它能检测突变点，并产生警告消息，例如：

Warning：discontinuity detected at A(2,3)

当出现这样的警告时，可以试着在另外的运算点处提取线性模型.

在[A，B，C，D]＝linmod2('systemname'，x，u，pert，Apert，Bpert，Cpert，Dpert)形式中，变量 Apert，Bpert，Cpert 和 Dpert 是为每一个状态和输入组合设定扰动水平的矩阵；因此，Apert 中第 ij 个元素是与获得的 A 矩阵中第 ij 个元素有关的扰动水平. 用下面的命令可以返回缺省的扰动水平：

[A，B，C，D，Apert，Bpert，Cpert，Dpert]＝linmod2('systemname'，x，u).

缺省情况下，系统时间被设为 0. 对于依赖时间的系统，可以将变量 pert 设为一个包含两个元素的向量，其中第二个元素用来设定获得线性模型的 t 值.

当要被线性化的模型本来就是一个线性模型时，截断误差将不存在；因此，可以将扰动水平设为任何值. 通常应该设为一个比较大的值，因为这可减小圆整误差. 此时，运算点不会对获得的线性模型有影响.

从非线性模型转换为线性模型时将保持状态的顺序. 对于 Simulink 的系统，可以用下面的命令获得与每一状态有关的包含有模块名字的字符串变量：

[sizes, x0, xstring]=sys

其中，xstring 是一个字符串向量，其第 i 行是与第 i 个状态有关的模块的名字. 模块图中输入和输出被依次编了号.

对于单输入多输出的系统，可以将它用 ss2tf 函数转换为传输函数，或者用 ss2zp 转换为零极点形式. 还可以用 ss 函数将线性化模型转换为 LTI 对象，函数 ss 产生的状态空间形式的 LTI 对象，还可以用 tf 或 zpk 进一步转换为传输函数或零点-极点-增益的形式.

5.5 动态系统平衡点分析(trim)

功能：确定动态系统的平衡点.

语法：[x, u, y, dx]=trim('systemname')

[x, u, y, dx]=trim('systemname', x0, u0, y0)

[x, u, y, dx]=trim('systemname', x0, u0, y0, ix, iu, iy)

[x, u, y, dx]=trim('systemname', x0, u0, y0, ix, iu, iy, dx0, idx)

[x, u, y, dx]=trim('systemname', x0, u0, y0, ix, iu, iy, dx0, idx, options)

[x, u, y, dx]=trim('systemname', x0, u0, y0, ix, iu, iy, dx0, idx, options, t)

[x, u, y, dx, options]=trim('systemname',...)

说明：从数学上讲，函数 trim 试图找到使状态导数为 0 的输入 u 和状态 x 的值. 这样的点被称为平衡点，在这些点系统处于稳定状态，它经常会出现在稳定状态的动态系统中. 函数 trim 从初始点开始，使用连续二次规划算法进行搜索，直到发现最接近平衡的点. 必须提供初始点. 如果 trim 不能找到平衡点，将返回搜索过程中遇到的状态导数最小最大意义上最接近于 0 的点，即返回从导数 0，最小化最大导数的点. 函数 trim 可以找到满足特定输入、输出和状态条件的平衡点，及系统以特定方式改变的点，即系统状态导数等于指定的非零值的点.

由于平衡点不是惟一的，因此需要确定状态 x，输入 u 和输出 y 的特定的值.

[x, u, y]=trim('systemname')：寻找最接近系统初始状态 x0 的平衡点. 找到的平衡点，是[x−x0, u, y]的绝对值的最小化最大值的点. 如果 trim 不能找到接近系统初始状态的平衡点，将返回系统最接近平衡的点. 特别地，它返回最小化 abs(dx−0)的点. 可以用下面的命令得到 x0：

[sizes, x0, xstr]=systemname ([], [], [], 0)

例 5.6 仍以图 5.6 的 linearmod 系统模型为例,输入命令:

```
[sizes,x0,xstr]=linearmod([],[],[],0)
sizes =
     3
     0
     2
     1
     0
     1
     1
x0 =
     0
     0
     0
xstr =
     'linearmod/反馈'
     'linearmod/变换'
     'linearmod/变换'
```

[x, u, y]=trim('systemname', x0, u0, y0);查找最接近 x0,u0,y0 的平衡点,即最小化 abs([x-x0; u-u0, y-y0])的最大值的点.

trim('systemname', x0, u0, y0, ix, iu, iy);查找最接近 x0,u0,y0 的平衡点,并满足一系列状态、输入、输出条件.整数向量 ix,iu 和 iy 选出 x0,u0 和 y0 中需要满足的元素值.如果不能找到严格满足指定的一系列条件的平衡点,将返回满足条件最接近的点,即,abs([x(ix)-x0(ix); u(iu)-u0(iu); y(iy)-y0(iy)]).

trim 使用一种有限制的优化方法,它限定状态导数为 0,并计算一个由 x,u 和 y 的期望值形成的最小最大值问题,对于这样的一个问题可能不存在可行的答案,如果是这样的话,trim 尽量使状态导数偏离 0 的值为最小.

[x, u, y, dx]=trim('systemname', x0, u0, y0, ix, iu, iy, dx0, idx);查找指定的非平衡点,即系统状态导数具有指定的非 0 值.其中,dx0 表示在搜索开始点指定的状态导数值,idx 是在 dx0 中选择的必须严格满足的元素的索引.

函数 trim 采用一个可选的变量 options,是一个优化参数的数组,是函数 trim 用来传递到优化函数中,用于查找平衡点的.优化函数依次使用这一数组来控制最优化过程,并返回过程信息.通过这种方法陈述潜在的优化过程,函数 trim 允许监视和微调平衡点的搜索.

有五个优化数组元素对平衡点搜索特别有用.表 5.2 给出了它们的值及说明.

例 5.7 对于一个如式(5.1)线性状态空间模型,一个名叫 systemname 的系统如图 5.11 所示.

表 5.2 五个重要的优化数组元素

序 号	缺省值	说 明
1	0	指定显示选项，0 指定不显示；1 指定列表输出；－1 禁止警告信息
2	0.0001	要终止搜索平衡点计算必须达到的精度
3	0.0001	要终止搜索，搜索目标函数必须达到的精度
4	0.0001	要终止搜索，状态导数必须达到的精度
10	N/A	返回用于平衡点搜索所使用的迭代数

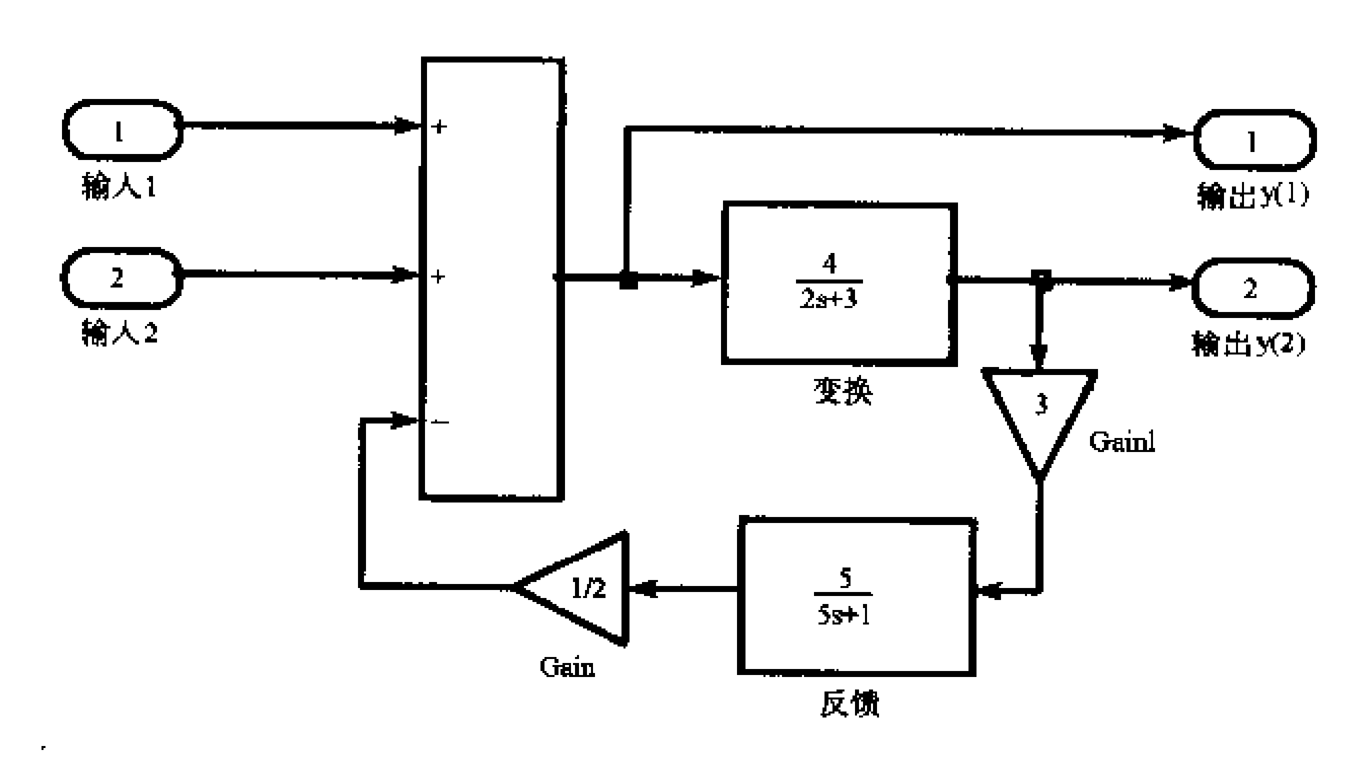


图 5.11 示例模型 systemname

(1) 提取线性模型

```
    [A,B,C,D]=linmod('systemname')
A =
     -0.2000      6.0000
     -0.5000     -1.5000
B =
      0     0
      1     1
C =
     -0.5000           0
           0      2.0000
D =
      1     1
      0     0
```

(2) Bode 相位、幅度与频率图

bode(A,B,C,D);

得到图 5.12 所示图形.

(3) 时间响应

```
step(A,B,C,D);
figure;
impulse(A,B,C,D);
```

得到图 5.13 和 5.14 所示图形.

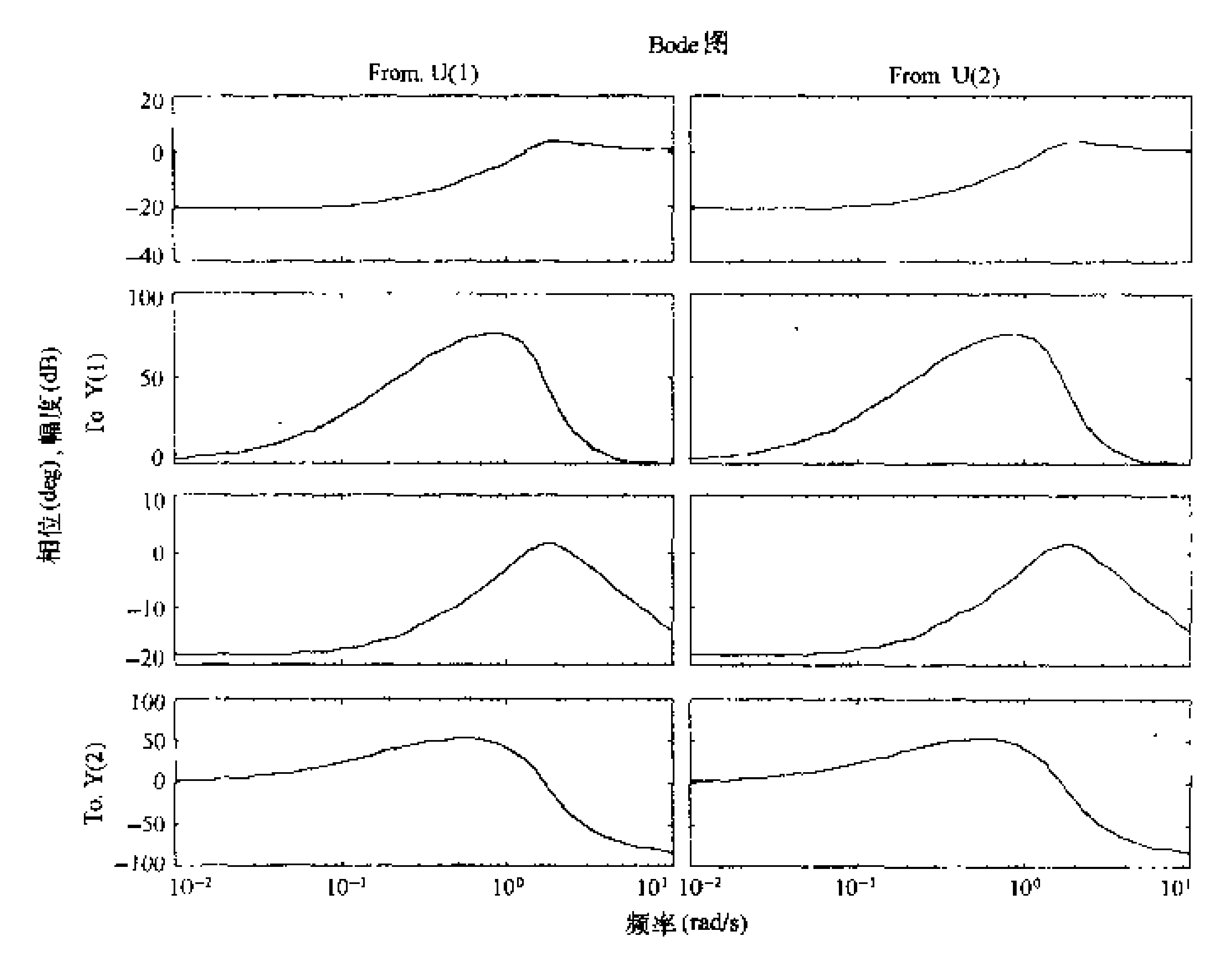


图 5.12　模型 systemname 系统线性化 Bode 相位、幅度与频率图

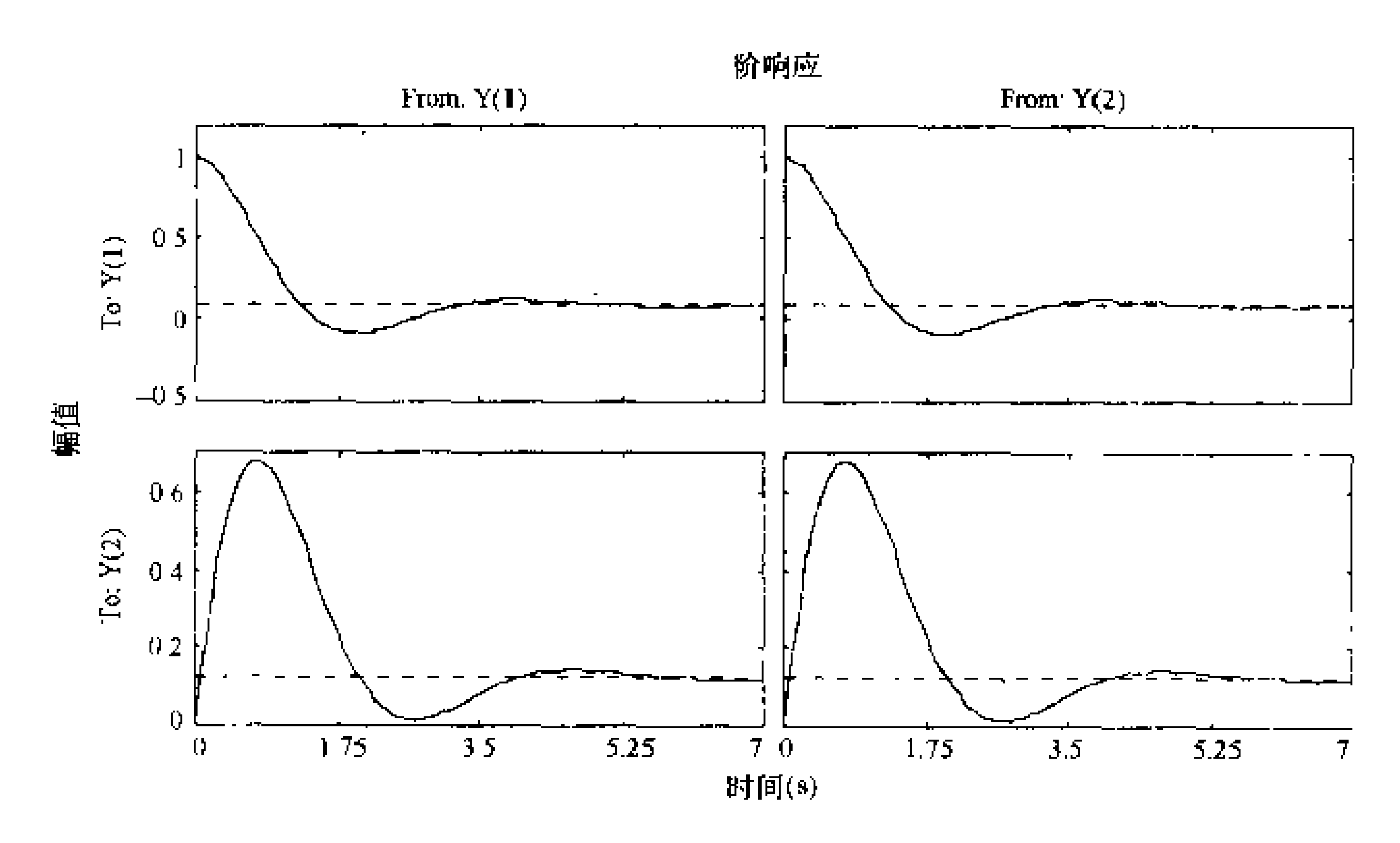


图 5.13　模型 systemname 系统线性化阶响应

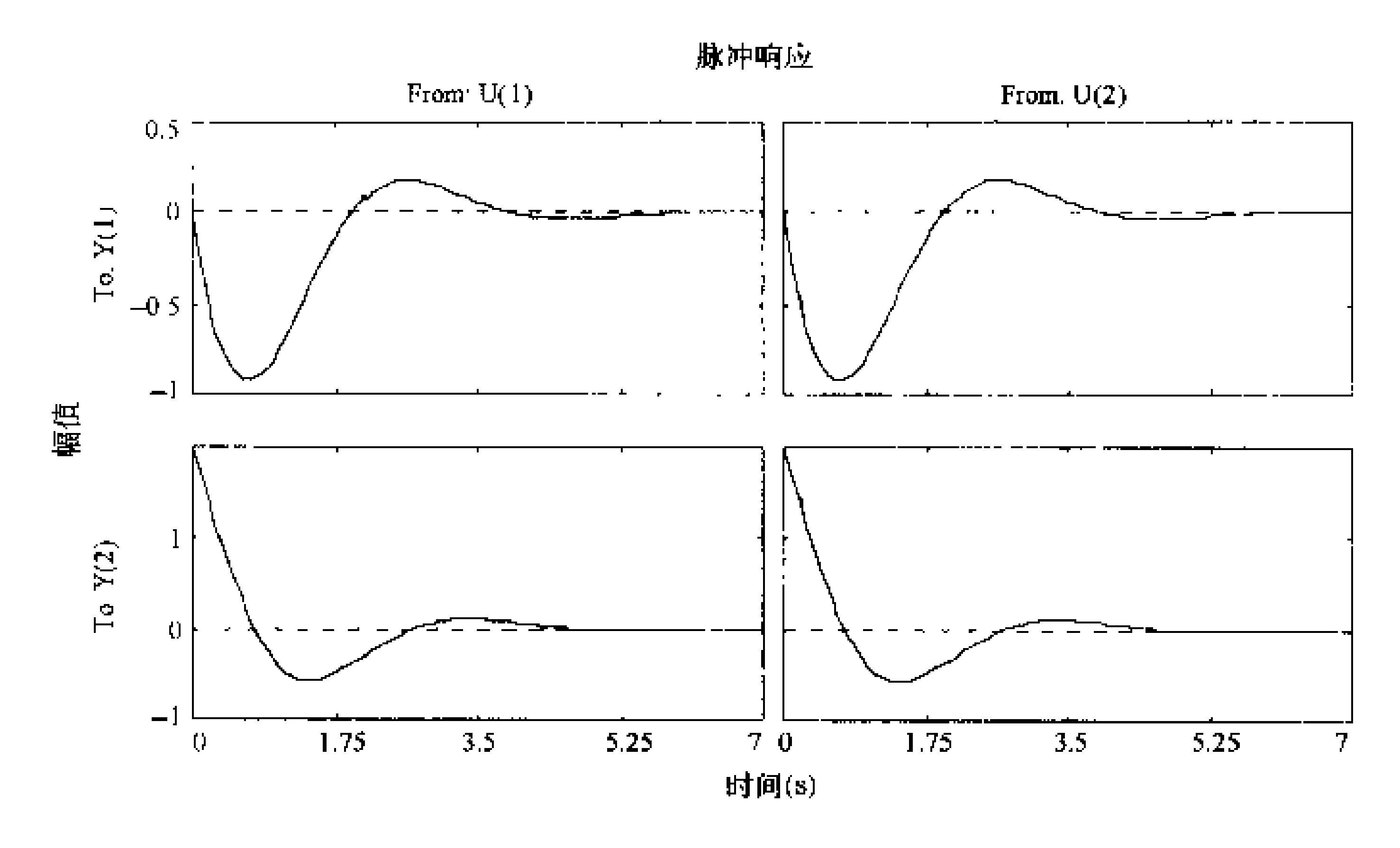


图 5.14　模型 systemname 系统线性化脉冲响应

(4)用下面的命令求平衡点

```
  [x,u,y,dx,options]=trim('systemname')
x =
     0
     0
u =
     0
     0
y =
     0
     0
dx =
     0
     0
options =
  Columns 1 through 7
        0    0.0001    0.0001    0.0000    0         0       1.0000
  Columns 8 through 14
        0         0    7.0000    2.0000    0    2.0000     500.0000
  Columns 15 through 18
        0    0.0000    0.1000    1.0000
```

迭代次数是：

```
  options(10)
ans =
```

```
    7
```

(5) 求一个在 x=[1; 1],u=[1; 1]附近的平衡点

```
  x0=[1; 1];
  u0=[1; 1];
  [x,u,y,dx,options]=trim('systemname',x0,u0)
x =
    1.9355
    0.0645
u =
    0.5323
    0.5323
y =
    0.0968
    0.1290
dx =
  1.0e-015 *
     0.0555
    -0.1110
options =
  Columns 1 through 7
         0    0.0001    0.0001    0.0000         0         0    1.0000
  Columns 8 through 14
    0.9355         0   25.0000    5.0000         0    2.0000  500.0000
  Columns 15 through 18
         0    0.0000    0.1000    1.0000
```

迭代次数是:

```
options(10)
ans =
    25
```

(6)求输出为 3 的平衡点

```
  y=[3; 3];
  iy=[1; 2];
  [x,u,y,dx,options]=trim('systemname',[],[],y,[],[],iy)
x =
    51.4286
     1.7143
u =
    14.1429
    14.1429
```

```
y =
     2.5714
     3.4286
dx =
   1.0e-014 *
   -0.3553
     0.4885
options =
  Columns 1 through 7
         0    0.0001    0.0001    0.0000         0         0    1.0000
  Columns 8 through 14
    0.4286         0   19.0000    4.0000         0    2.0000  500.0000
  Columns 15 through 18
         0    0.0000    0.1000    1.0000
```

(7)求输出为4,导数设为0和1的平衡点

```
  y=[4; 4];
  iy=[1; 2];
  dx=[0;1];
  idx=[1;2];
  [x,u,y,dx,options]=trim('systemname',[],[],y,[],[],iy,dx,idx)
x =
    60.0000
     2.0000
u =
    17.0000
    17.0000
y =
     4.0000
     4.0000
dx =
     0.0000
     1.0000
options =
  Columns 1 through 7
         0    0.0001    0.0001    0.0000         0         0    1.0000
  Columns 8 through 14
    0.0000         0   25.0000    5.0000         0    2.0000  500.0000
  Columns 15 through 18
         0    0.0000    0.1000    1.0000
```

迭代次数是：

```
options(10)
ans =
    25
```

尽管从给定的初始点开始，找到了稳定状态的平衡点，但只是一局部的值.有可能还有更稳定的 x，u 和 y 的值存在，因为无法保证它是一个全局最优解，除非该最优化问题是单调变化的.因此，要寻求全局的解，必须设定 x，u 和 y 的多组初始估计值分别试一下.对于有间断点的系统，trim 不能正常工作.

第六章　MATLAB 仿真模块库

6.1　MATLAB 仿真模块库简介

在 MATLAB 命令行状态下，运行 simulink 命令，则显示 MATLAB 中所安装的所有仿真模块库(集)，如图 6.1 所示.

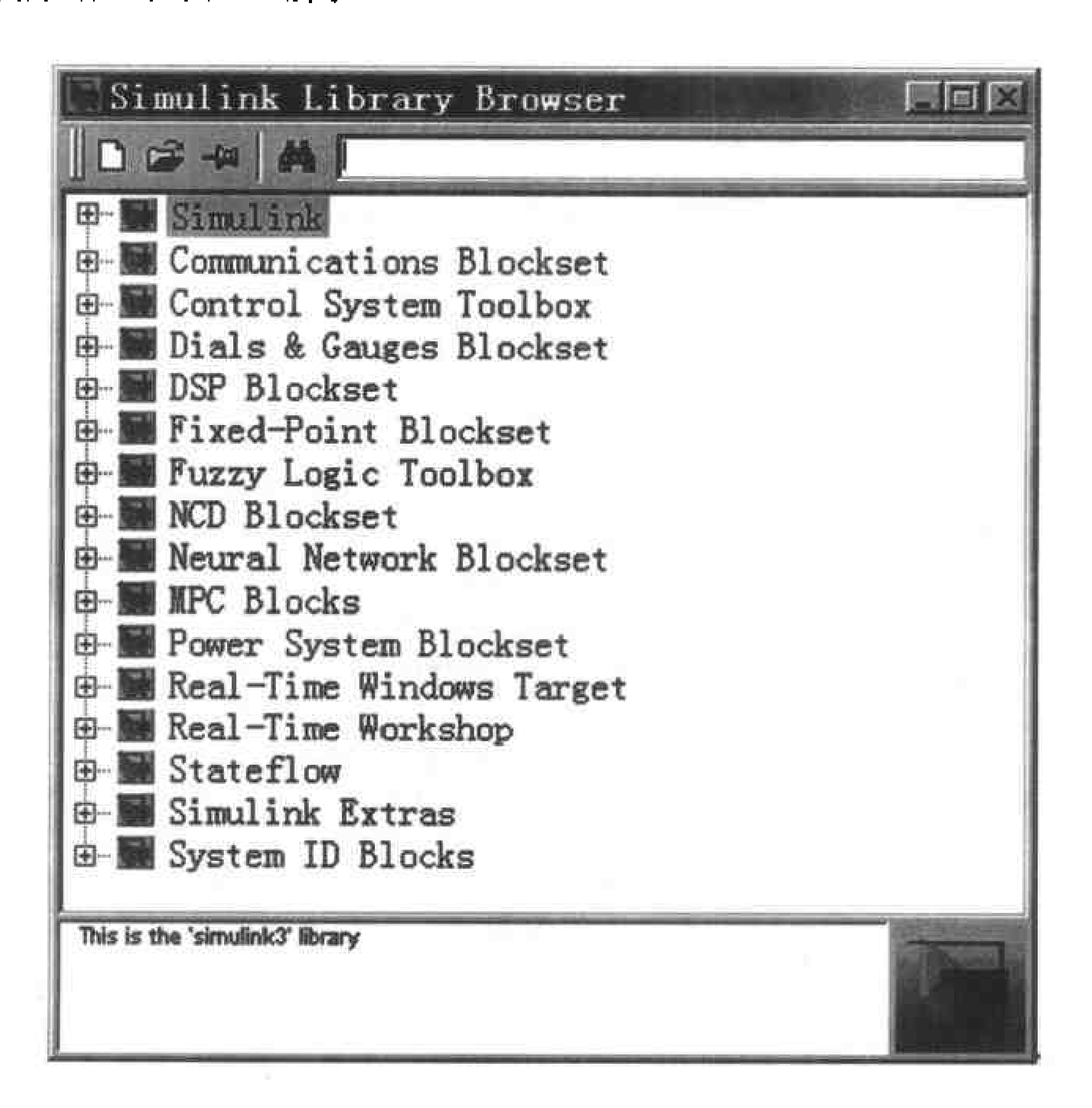


图 6.1　在 Simulink 库浏览窗口显示 MATLAB 仿真模块库(集)

6.2　Simulink 库

打开 Simulink 库，可显示如图 6.2 所示的模块库图. 其中包含的模块库有：

1) Continuous(连续库)，包含有发生信号的一些模块. 打开该库后显示如图 6.3 所示的模块图.

2) Discrete(离散库)，打开该库，其中模块如图 6.4 所示.

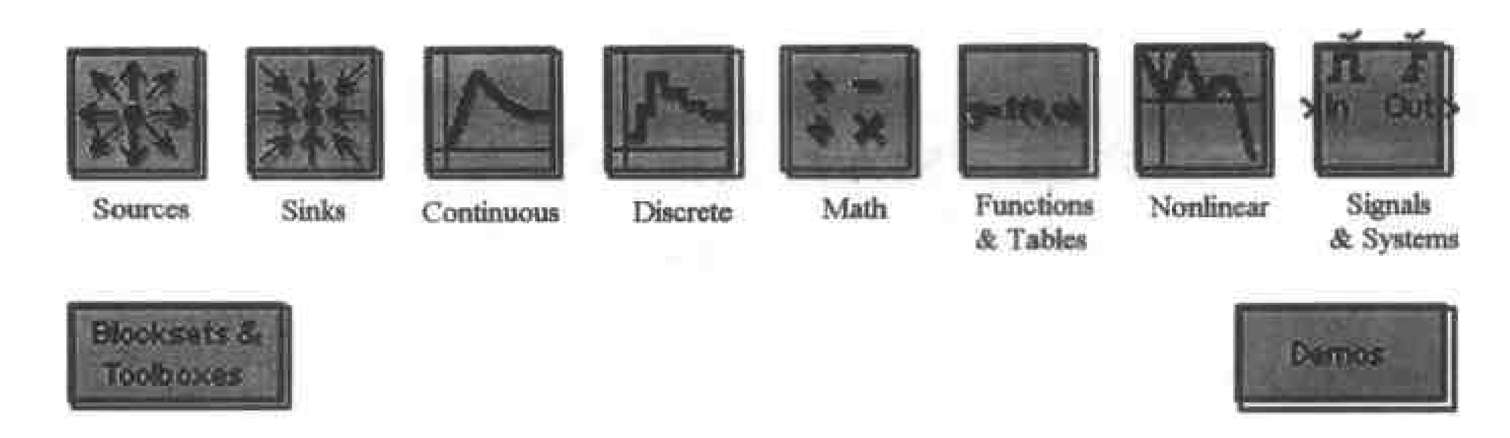


图 6.2 Simulink 库中的模块库

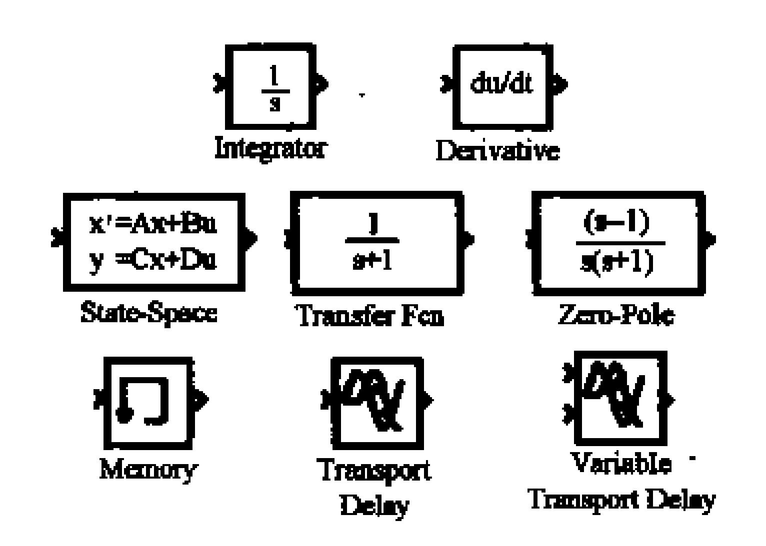


图 6.3 连续库中的模块图

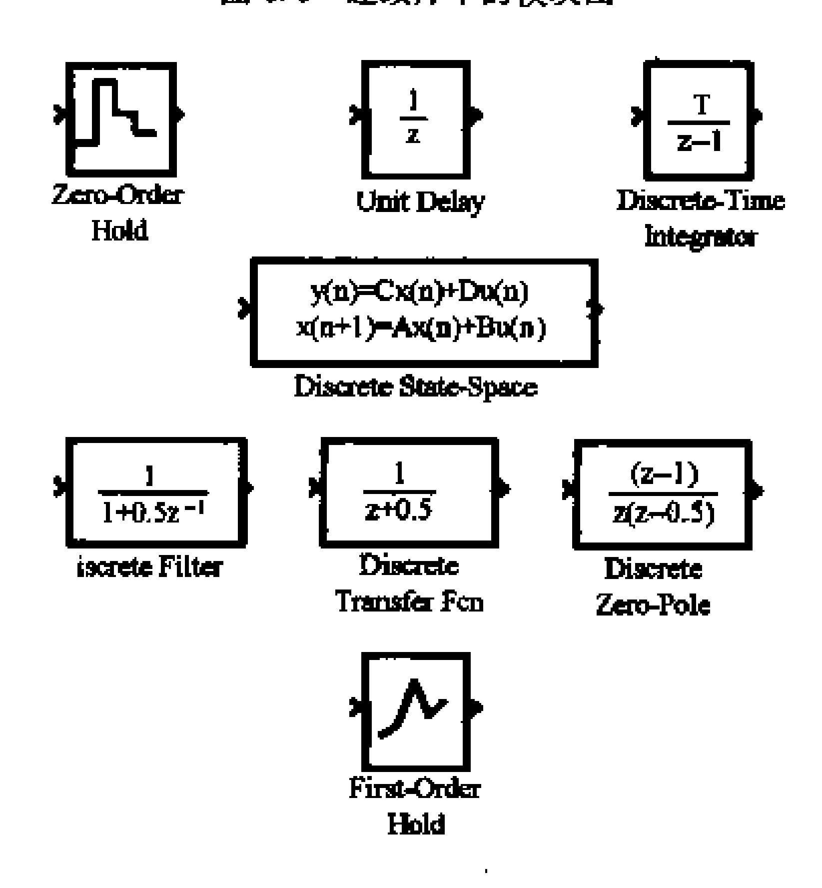


图 6.4 离散库中的模块图

3) Functions & Tables(函数与表库),打开该库,显示如图 6.5 所示的模块图.

4) Math(数学运算库),打开该库,显示的数学运算模块如图 6.6 所示.

5) Nonlinear(非线性库),打开该库,显示的非线性模块如图 6.7 所示.

6) Sinks(接收库),打开该库,显示如图 6.8 所示的模块图.

7) Signals & Systems(信号与系统库),打开该库,显示如图 6.9 所示的模块图.

8) Source(源库),打开源库,显示图 6.10 所示的模块图.

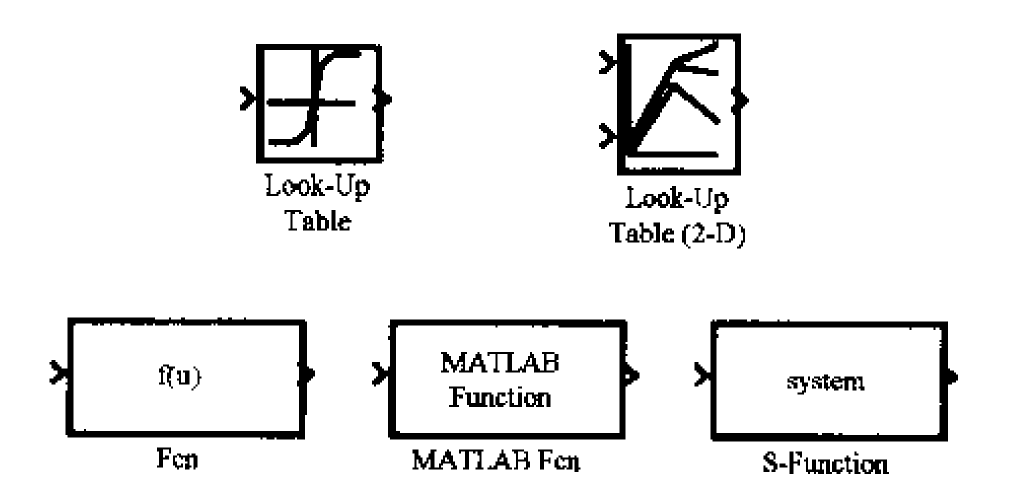


图 6.5 函数与表库中的模块图

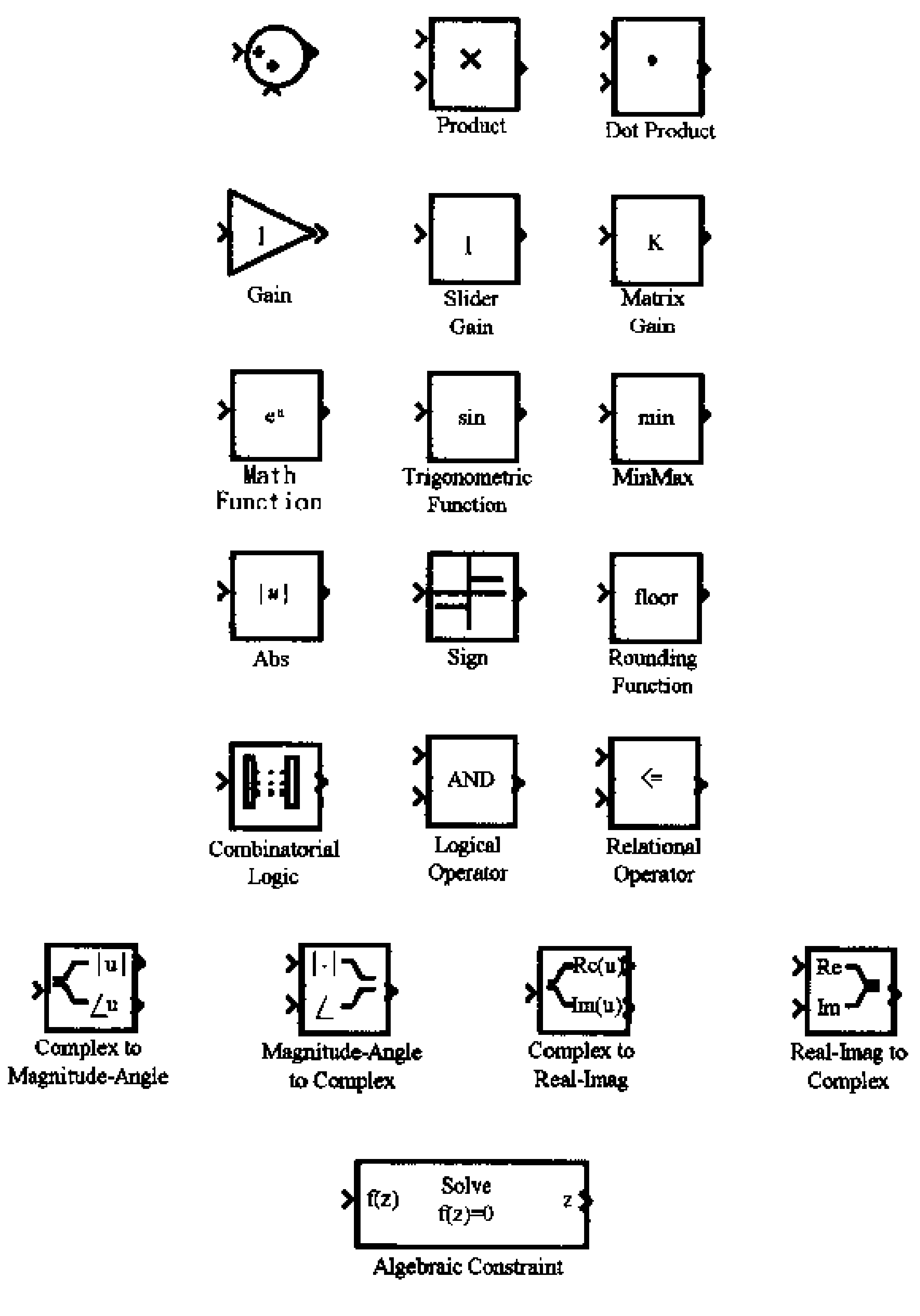


图 6.6 数学运算库中的模块图

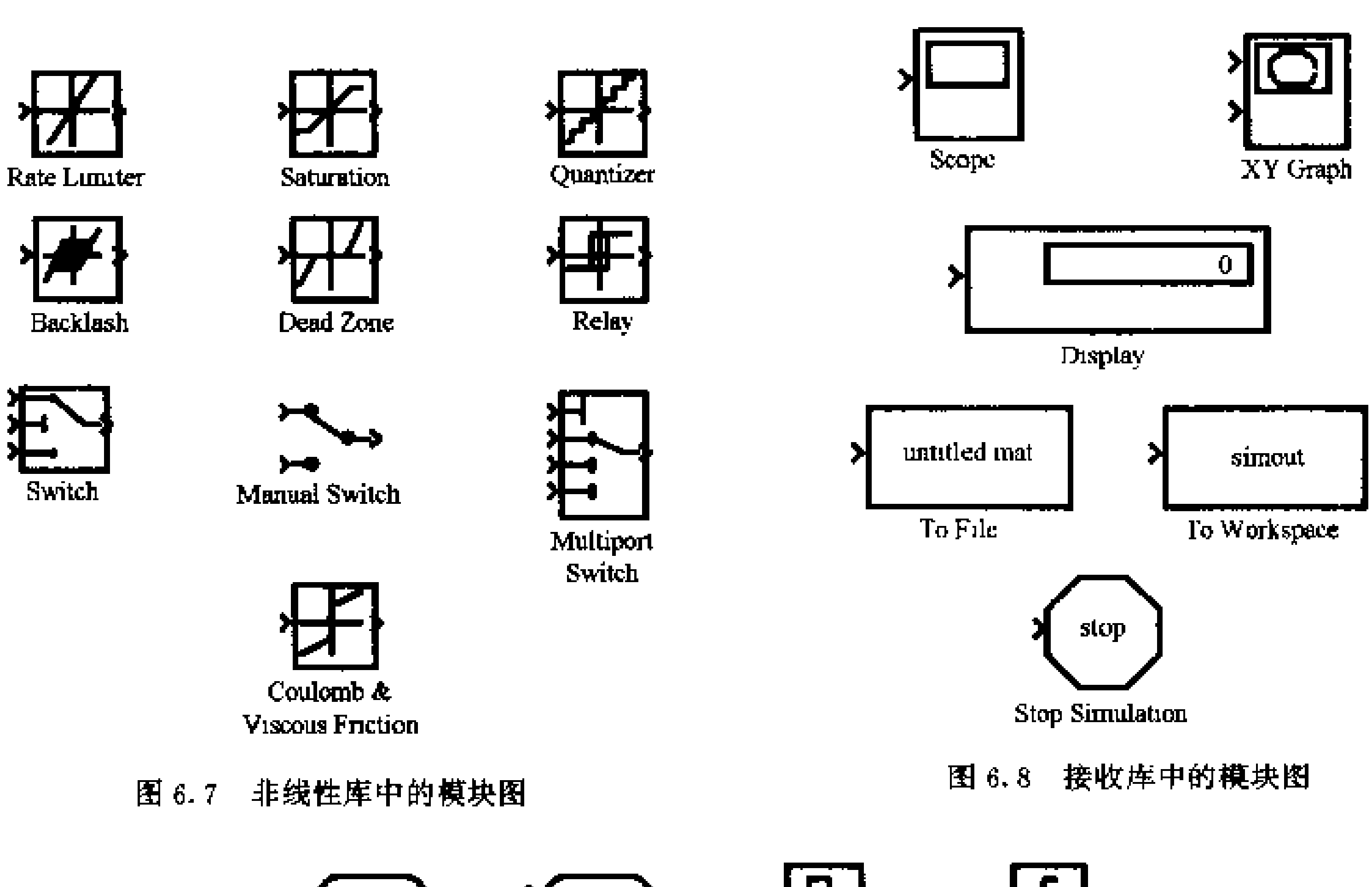


图 6.7　非线性库中的模块图

图 6.8　接收库中的模块图

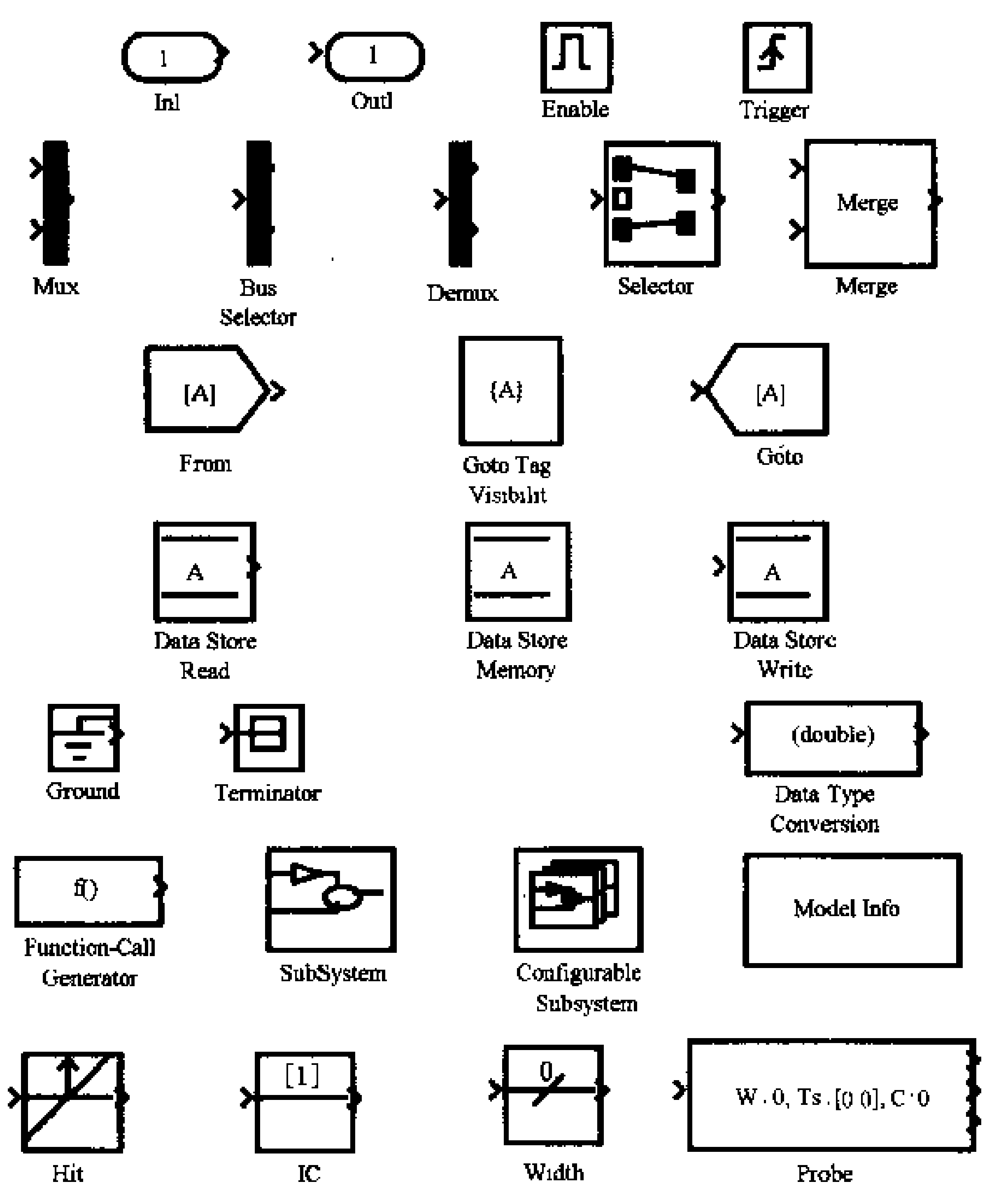


图 6.9　信号与系统库中的模块图

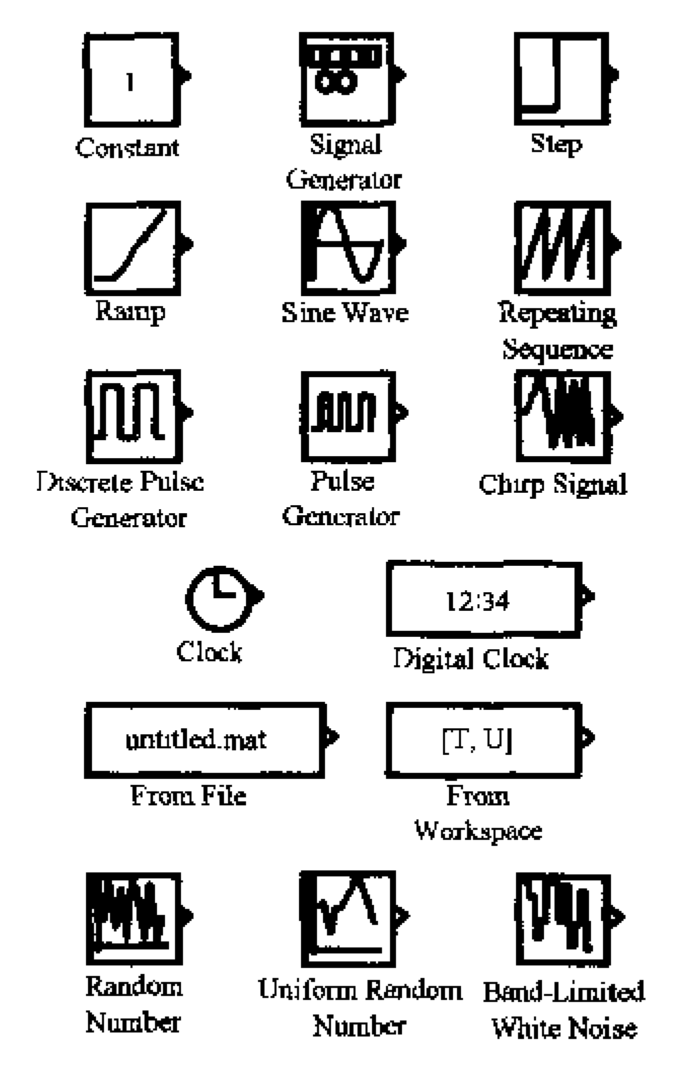


图 6.10 源库中的模块图

6.3 Communications Blockset(通信模块集)

打开通信模块集,显示图 6.11 所示的模块库图.

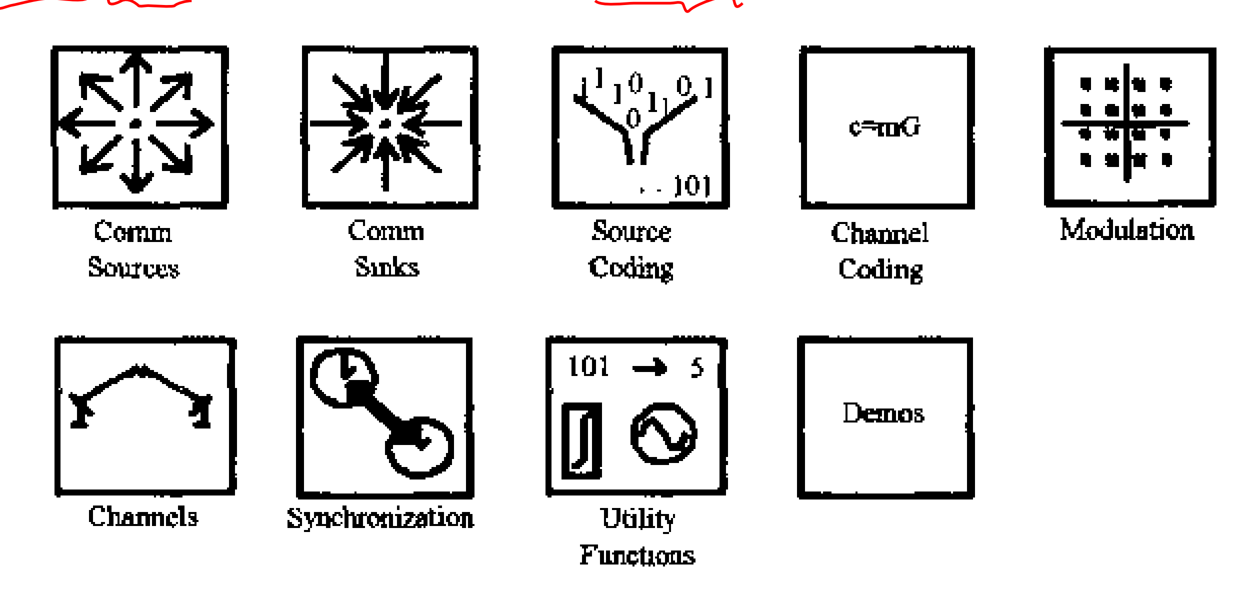


图 6.11 通信模块集中的模块库

1) Channel Coding(信道编码库),该库中又包括的有:模块编码库和卷积编码库.

模块编码库中包含有各种编码和解码成对模块以及演示模块.

(a) 线性编码模块组. 有:二进制向量线性编码、解码和演示三个模块,二进制序列线性编码、解码和演示三个模块.

(b) 循环编码模块组. 有:二进制向量循环编码、解码和演示三个模块,二进制序列循环编码、解码和演示三个模块.

(c) Hamming 编码模块组. 有:二进制向量 Hamming 编码、解码和演示三个模块,二

进制序列 Hamming 编码、解码和演示三个模块.

(d) BCH 编码模块组. 有:二进制向量 BCH 编码、解码和演示三个模块,二进制序列 BCH 编码、解码和演示三个模块.

(e) Reed-Solomon 编码模块组. 有:整数向量 RS 编码、解码和演示三个模块,二进制向量 RS 编码、解码和演示三个模块,整数序列 RS 编码、解码和演示三个模块,二进制序列 RS 编码、解码和演示三个模块.

卷积编码库中包含有:卷积编码、Viterbi 解码和演示三个模块.

2) Channels(信道库),包含的模块有:

(a) 加零均值 Gauss 白噪声(AWGN)信道模块及四个演示模块.

(b) 加二进制误差信道模块及演示模块.

(c) 有限二进制误差模块及演示模块.

(d) 定参数 Rayleigh 衰减信道模块,变参数 Rayleigh 衰减信道模块及演示模块.

(e) 定参数加 Rician 噪声信道模块,变参数加 Rician 噪声信道模块,及两个演示模块.

3) Comm Sinks(通信接收库),包含的模块有:

(a) 触发写文件模块及触发文件 I/O 演示模块.

(b) 触发眼孔图样/散布图模块及演示模块.

(c) 采样时间眼孔图样/散布图模块及演示模块.

(d) 误差率计算模块及演示模块.

4) Comm Sources(通信源库),包含的模块有:

(a) 触发文件读入模块及触发文件 I/O 演示模块.

(b) 采样读工作空间变量模块,具有同步脉冲的采样读工作空间变量模块.

(c) 具有采样率的向量脉冲模块.

(d) 伪随机序列发生器模块及演示模块.

(e) 均匀分布噪声发生器模块及演示模块.

(f) Gauss 分布噪声发生器模块及演示模块.

(g) 随机整数发生器模块及均匀分布整数演示模块.

(h) Poisson 分布随机整数发生器模块及演示模块.

(i) 二进制向量发生器模块及演示模块.

(j) Bernoulli 分布随机数发生器模块及演示模块.

(k) Rayleigh 分布噪声发生器模块及演示模块.

(l) Rician 分布噪声发生器模块及演示模块.

5) Modulation(调制库),调制库中包含四个模块库,它们是:数字基带调制模块库,数字通带调制模块库,模拟基带调制模块库,模拟通带调制模块库.

数字基带调制模块库,包含的模块有:

(a) 基带 MASK (Multiple Amplitude Shift Keying,多幅移键法)调制、解调及演示三个模块.

(b) 基带 S-QASK(Quadrature Amplitude Shift Keying,正交振幅相移键控法)调制、解调及演示三个模块.

(c) 基带 A-QASK 调制、解调及演示三个模块.

(d) 基带 MFSK(Multiple Frequency Shift Keying,多频移键控法)调制模块,基带相

干 MFSK 解调模块,基带非相干 MFSK 解调模块,及演示四个模块.

(e) 基带 MPSK(Multiple Phase Shift Keying,多相移键控法)调制、解调及演示三个模块.

数字通带调制模块库,包含的模块有:

(a) 通带 MASK 调制、解调及演示三个模块.

(b) 通带 S-QASK 调制、解调及演示四个模块.

(c) 通带 A-QASK 调制、解调及演示三个模块.

(d) 通带 MFSK 调制模块,通带相干 MFSK 解调模块,通带非相干 MFSK 解调模块,及演示四个模块.

(e) 通带 MPSK 调制、解调及演示三个模块.

(f) 通带 DPSK(Differential Phase Shift Keying,差分相移键控法)调制、解调两个模块.

(g) 基带 MSK(Minimun Phase Shift Keying, 最小相移键控法)调制、解调两个模块.

(h) 通带 OQPSK(Offset Quadrature Phase Shift Keying,偏移正交相移键控法)调制、解调两个模块.

模拟基带调制模块库,包含的模块有:

(a) 基带 DSB-SC(Double Side Band Shift Control,双边频带移位控制) AM(Amplitude Modulation,调幅)、解调及演示三个模块.

(b) 基带 QAM(Quadrature Amplitude Modulation,正交幅度调制)、解调及演示三个模块.

(c) 基带 FM(Frequency Modulation,调频)、解调及演示三个模块.

(d) 基带 PM(Phase Modulation,相位调制)、解调及演示三个模块.

(e) 基带 SSB-AM(Single Side Band Amplitude Modulation,单边带调幅)、解调及演示三个模块.

(h) 具有传输载波的基带 AM、解调及演示三个模块.

模拟通带调制模块库.包含的模块有:

(a) 通带 DSB-SC AM、解调及演示三个模块.

(b) 通带 QAM、解调及演示三个模块.

(c) 通带 FM、解调及演示三个模块.

(d) 通带 PM、解调及演示三个模块.

(e) 通带 SSB-AM、解调及演示三个模块.

(f) 具有传输载波的通带 AM、解调及演示三个模块.

6) Source Coding(源编码库),该库中包含的模块有:

(a) 标量量化编码、解码及演示三个模块.

(b) 激活量化编码及演示两个模块.

(c) DPCM(Differential Pulse Code Modulation,差分脉码调制技术)编码、解码及演示三个模块.

(d) μ 规则压缩、解压两个模块.

(e) A 规则压缩、解压两个模块.

7) Synchronization(同步库),同步库中包含的模块有:

(a) PLL(Phase Locked Loop,相同步回路, 锁相回路)模块,基带 PLL 模型模块及演

示模块.

(b) 进料泵 PLL 模块.

(c) 线性化基带 PLL 模块.

8) Utility Functions(实用函数库),实用函数库包含的模块有:

(a) 离散时间模积分器模块.

(b) 模积分器模块.

(c) 离散 VCO(Voltage Controlled Oscillator,压控振荡器)模块.

(d) VCO(Voltage Controlled Oscillator,压控振荡器)模块.

(e) 可复位数值计数器模块.

(f) 错误计数器模块.

(g) 数据绘图器及演示两模块.

(h) 二进制微分编码器和解码器两模块.

(i) 窗口积分器模块.

(j) 包络检测器模块.

(k) 十进制整数标量到向量转换器模块.

(l) 交错模块及两个演示模块.

(m) 预定复位积分模块.

(n) 信号边沿检测模块.

(o) 十进制整数向量到标量转换器模块.

(p) 扰频器、解扰器及演示三个模块.

(q) 寄存器移位及演示两个模块.

(r) 触发缓冲器模块.

(s) 触发向量信号重新分布及演示两个模块.

(t) 向量信号重新分布及演示两个模块.

6.4 Control System Toolbox(控制系统工具箱)

打开控制系统工具箱,显示控制系统工具箱中使用的 Simulink 模块,如图 6.12 所示. 包括:输入点、输出点和 LTI 模块.

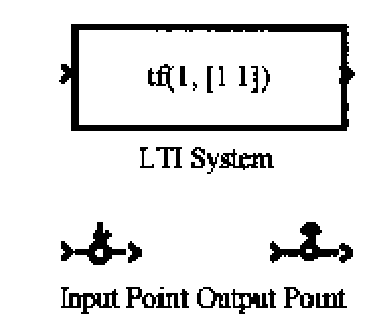


图 6.12 控制系统工具箱中使用的 Simulink 模块

6.5 Dials & Gauges Blockset(面板和仪表模块集)

包含的模块库有:Dashboard-Based Instrumentation(基于面板的仪器库),Model-

Based Instrumentation(基于模型的仪器库).

6.6 DSP Blockset(数字信号处理模块集)

打开 DSP 模块集,显示图 6.13 所示的模块库图.包含的模块库有:

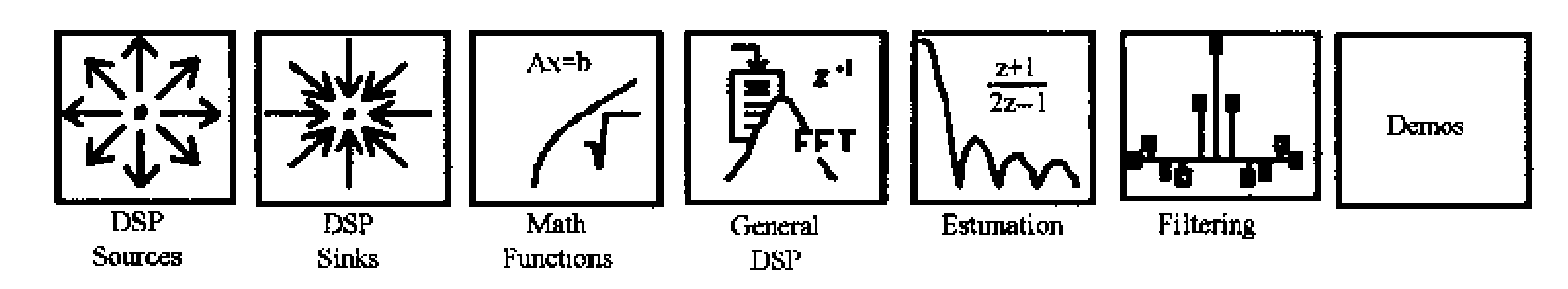


图 6.13 DSP 模块集中的模块库

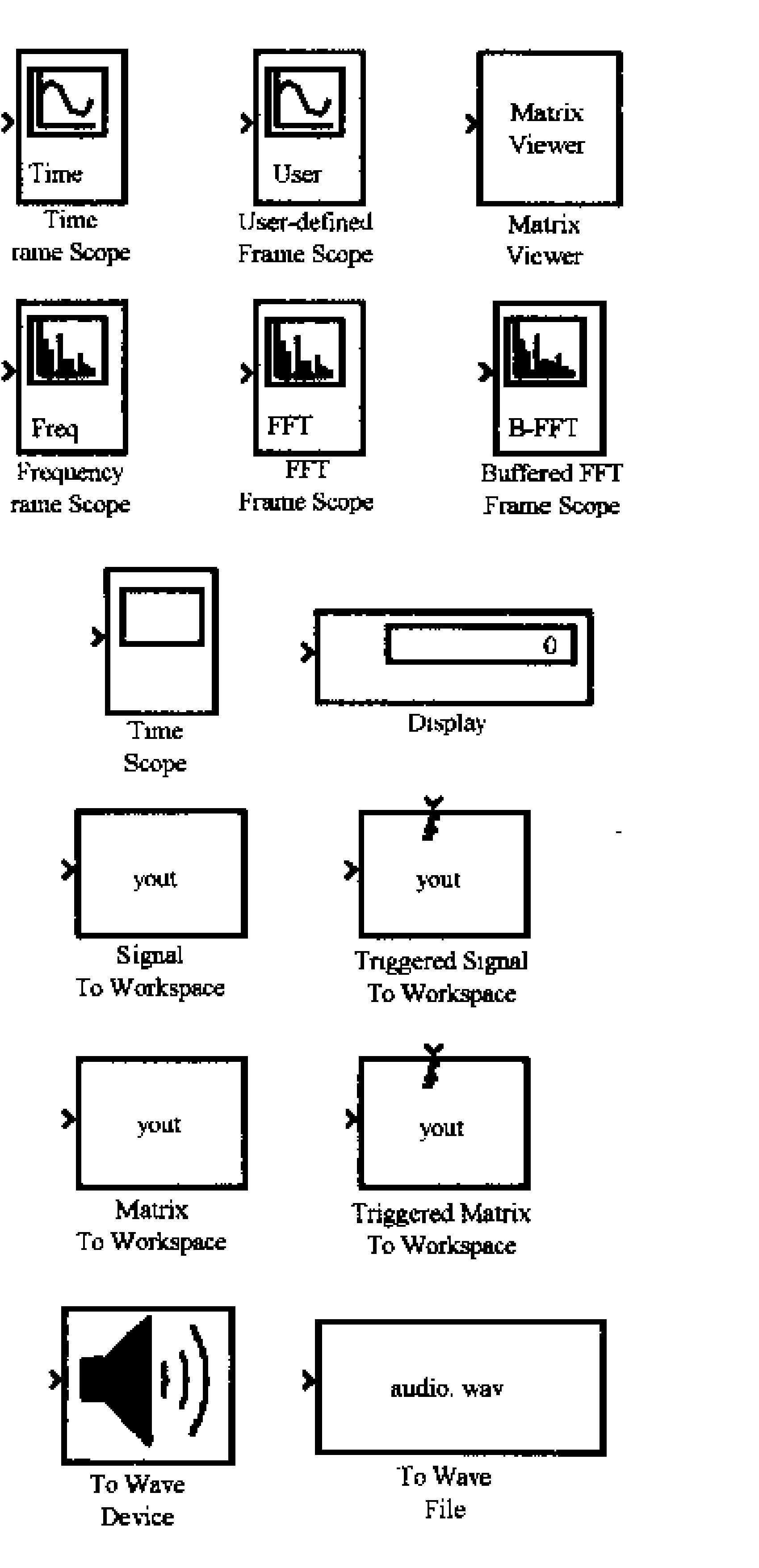


图 6.14 DSP 接收模块库中的模块

1）DSP Sinks（DSP 接收库），该库中包含的模块如图 6.14 所示.

2）DSP Sources（DSP 源库），该库中包含的模块如图 6.15 所示.

3）Estimation（估计库），估计库中包含有参数估计模块库和功率谱估计模块库.其中参数估计模块库包含：Yule-Walker AR 估计模块，Burg AR 估计模块，协方差 AR 估计模块，改进协方差 AR 估计模块.

功率谱估计模块库包含：短时 FFT 模块，幅度 FFT 模块，Yule-Walker AR 谱估计模块，Burg AR 谱估计模块，协方差 AR 谱估计模块，改进协方差 AR 谱估计模块.

4）Filtering（滤波器库），滤波器库包含有：滤波器设计模块库，滤波器实现模块库，自适应滤波器模块库，多级滤波器模块库.

滤波器设计模块库包含的模块如图 6.16 所示.

滤波器实现模块库包含的模块如图 6.17 所示.

自适应滤波器模块库，包含的模块如图 6.18 所示.

多级滤波器模块库，包含的模块如图 6.19 所示.

5）General DSP（通用 DSP 库），该库中包含有：信号操作、信号变换、信号缓冲、开关

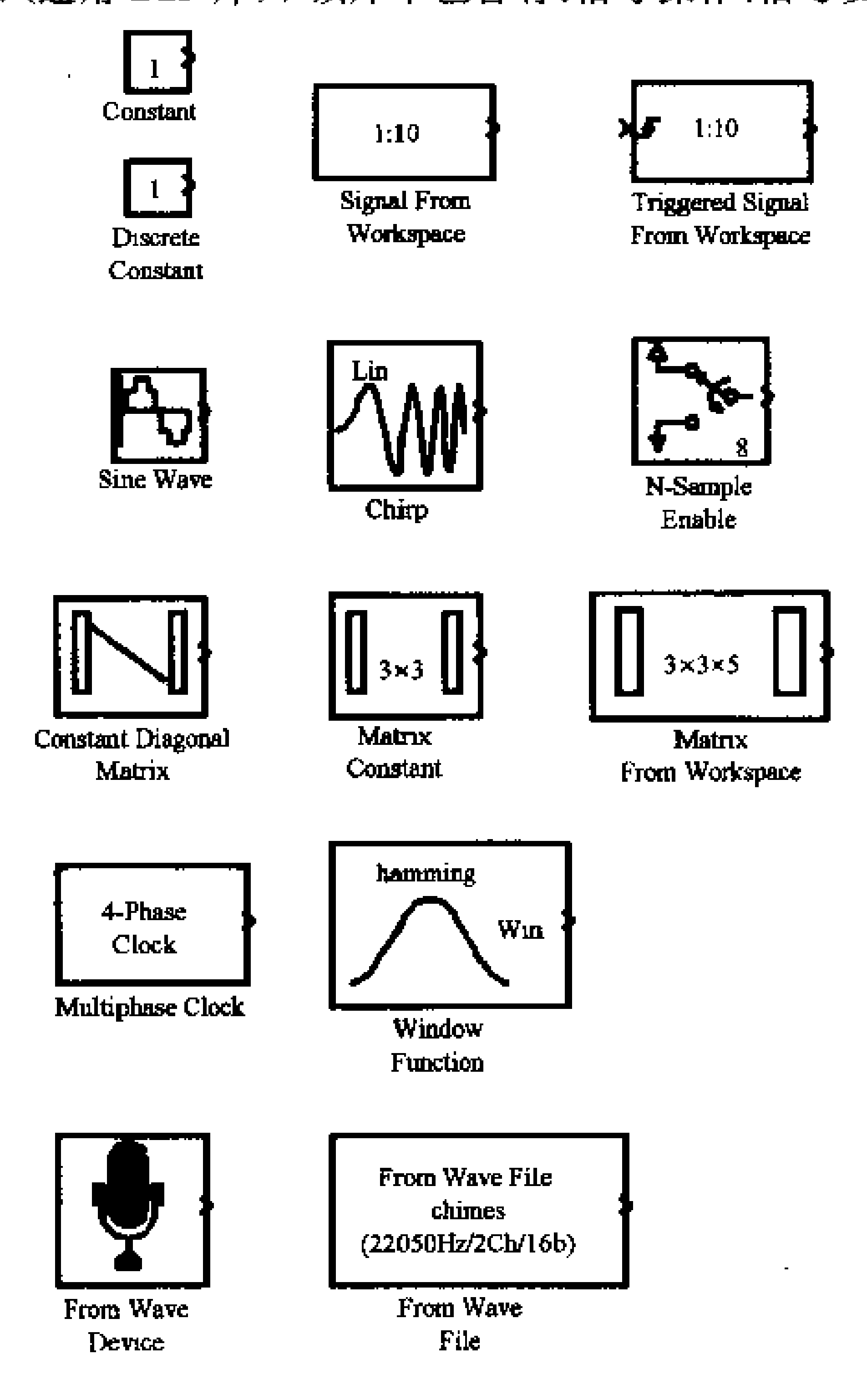


图 6.15 DSP 源库中包含的模块

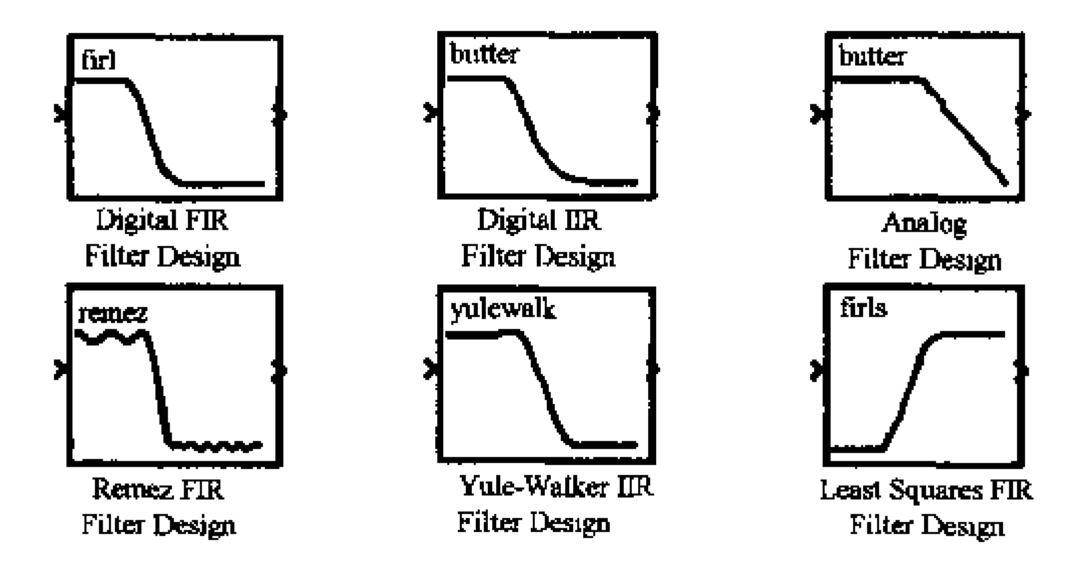


图 6.16 滤波器设计模块库中的模块

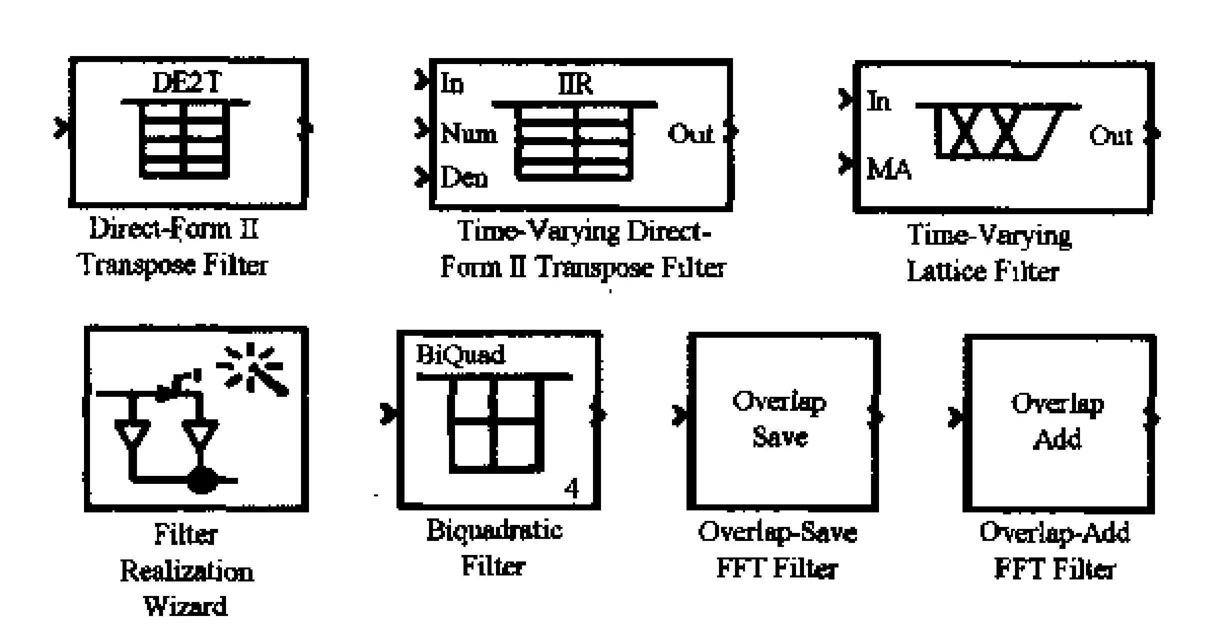


图 6.17 滤波器实现模块库包含的模块

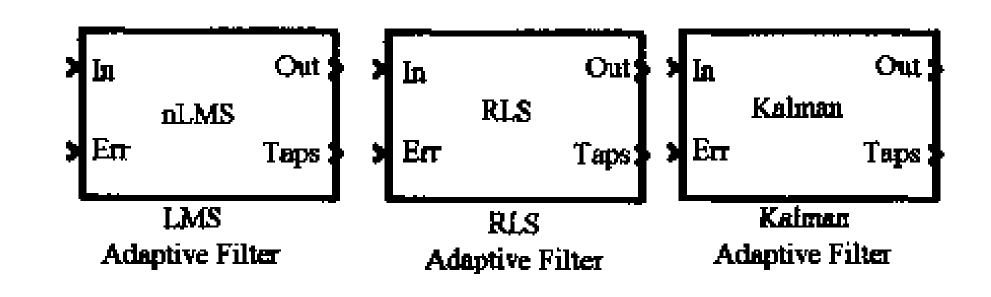


图 6.18 自适应模块库中的模块

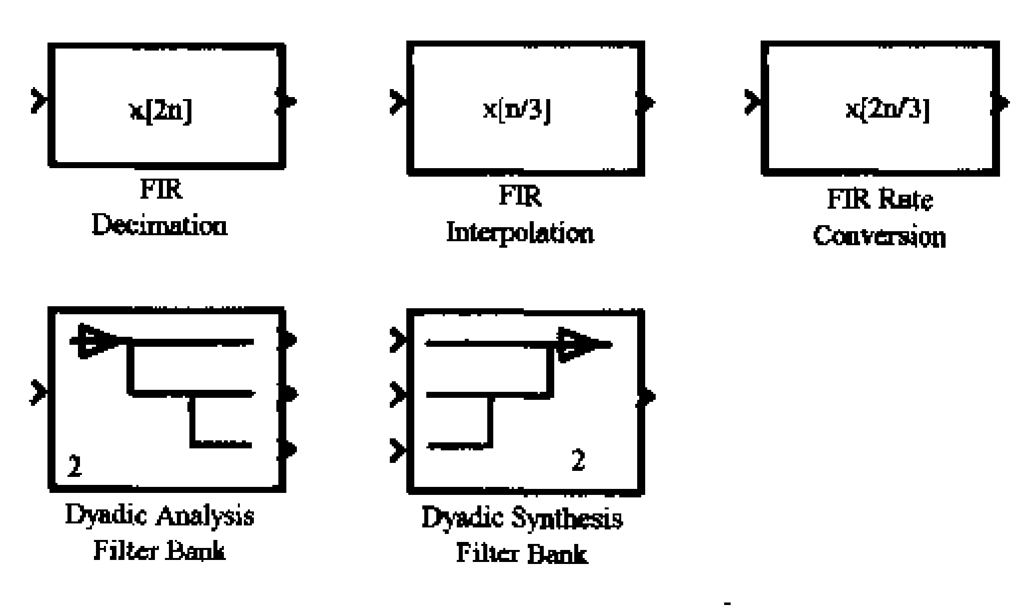


图 6.19 多级滤波器模块库中的模块

与计数器四个模块库，如图 6.20 所示. 各模块库中的模块如图 6.21 至 6.24 所示.

6) Math Functions(数学函数库)，该库中包含有基本函数、向量函数、矩阵函数、线性代数和统计五个模块库，如图 6.25 所示. 各模块库中的模块如图 6.26 至 6.30 所示.

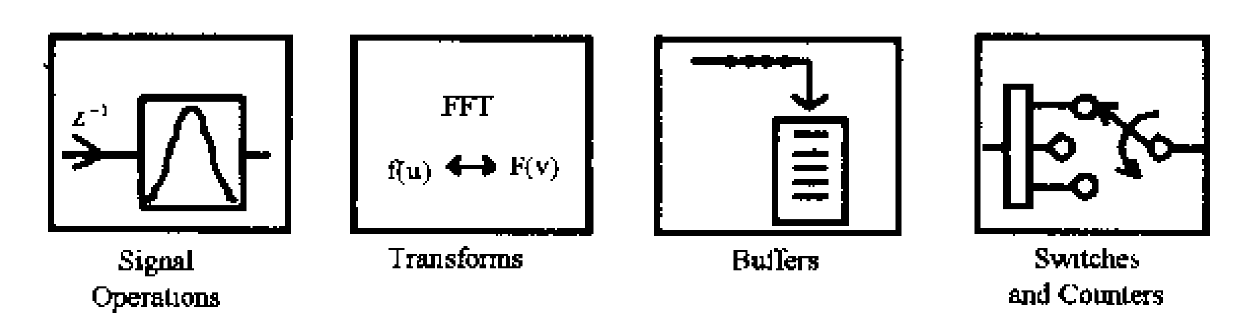


图 6.20 通用 DSP 库中的模块库

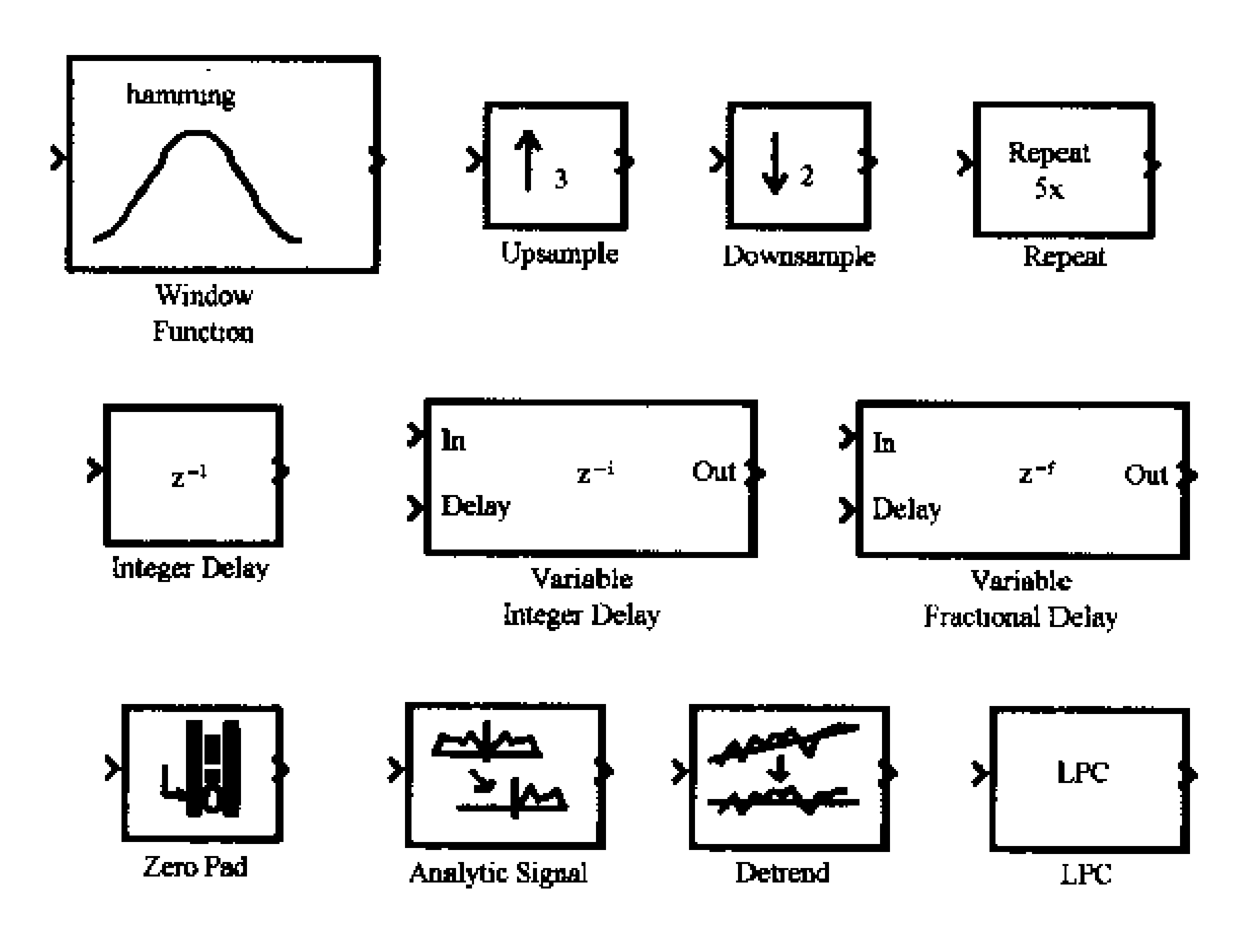


图 6.21 信号操作模块库中的模块

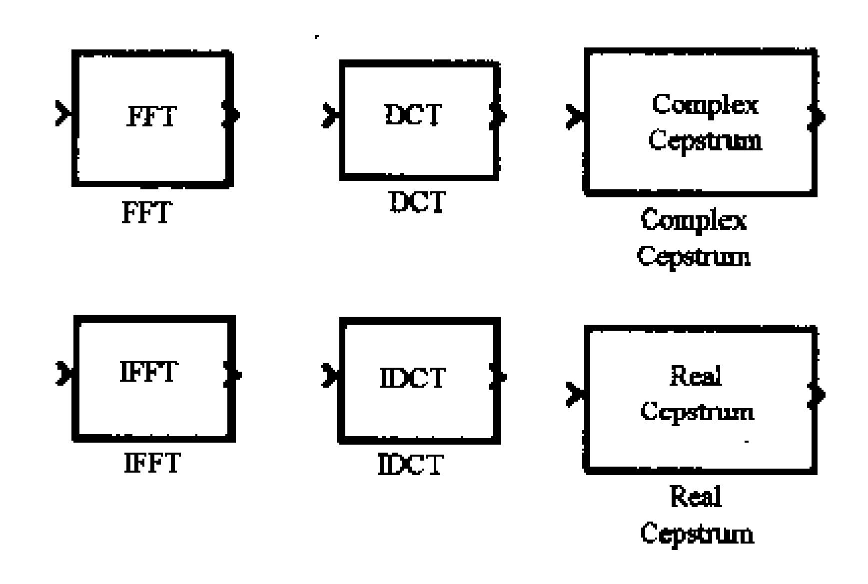


图 6.22 信号变换模块库中的模块

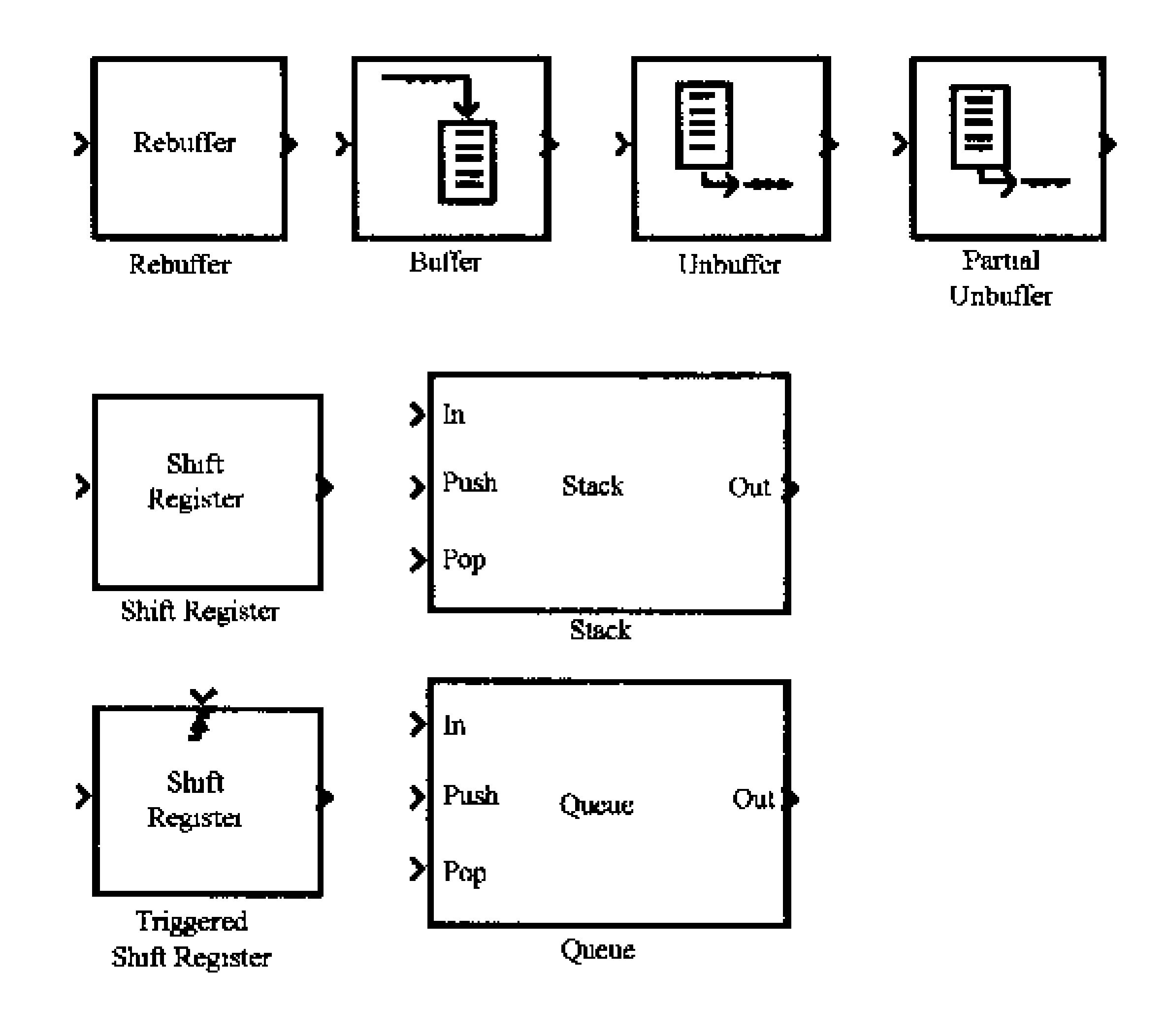


图 6.23 信号缓冲模块库中的模块

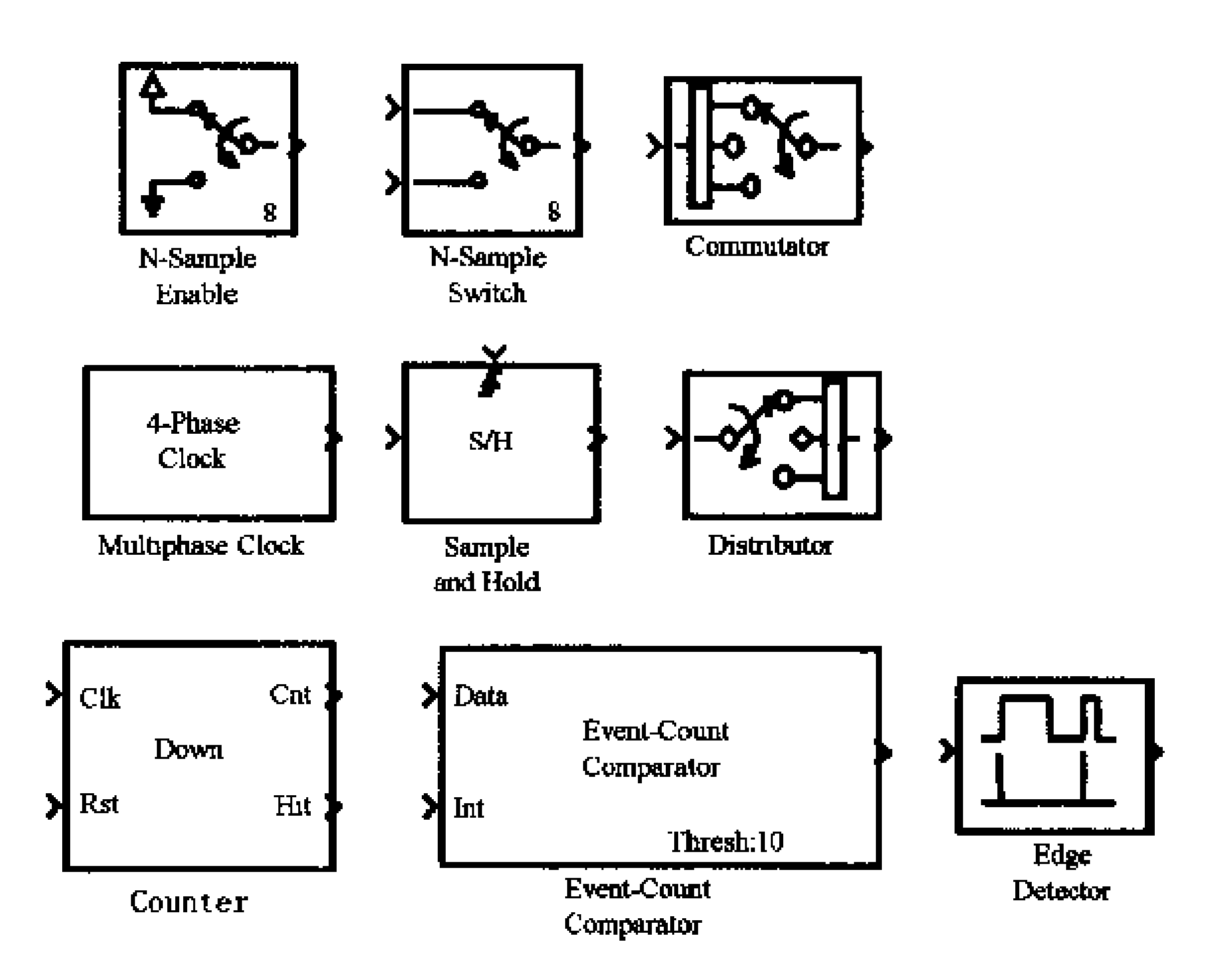


图 6.24 开关与计数器模块库中的模块

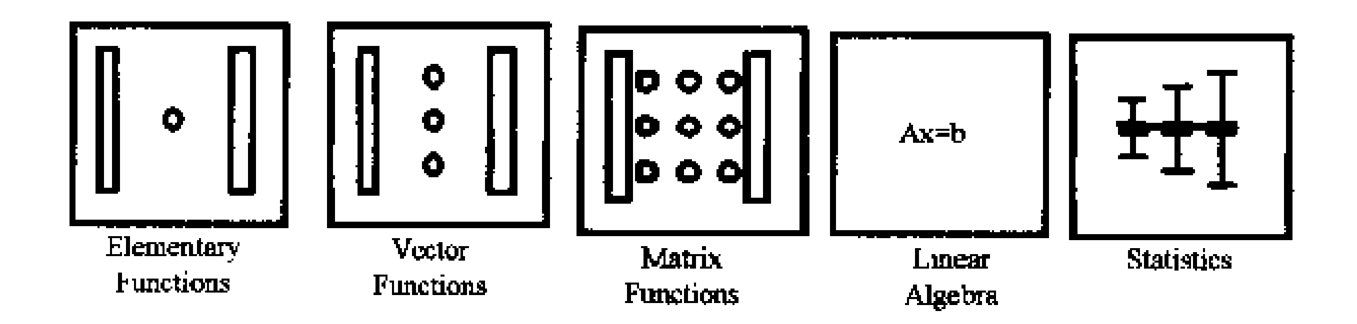


图 6.25　数学函数库中的模块库

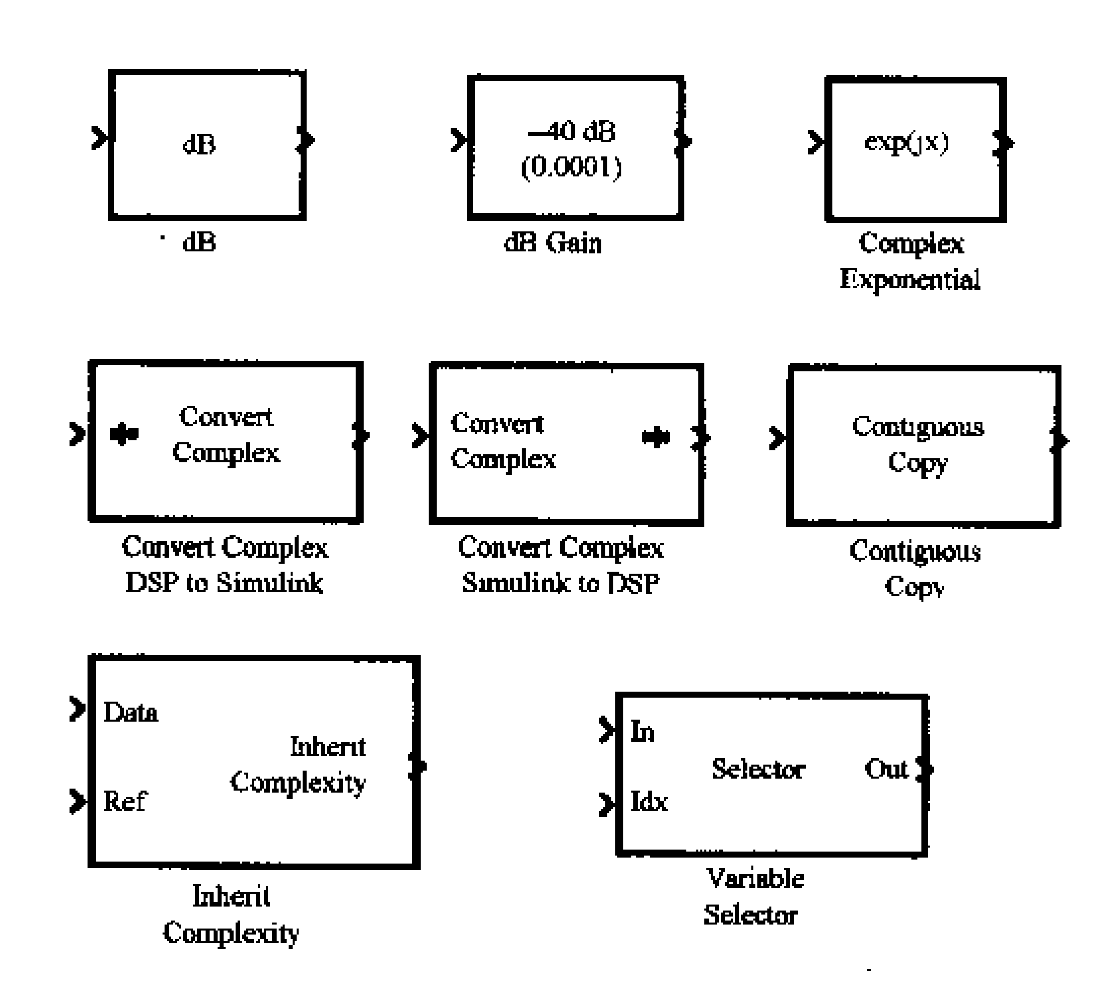


图 6.26　基本函数模块库中的模块

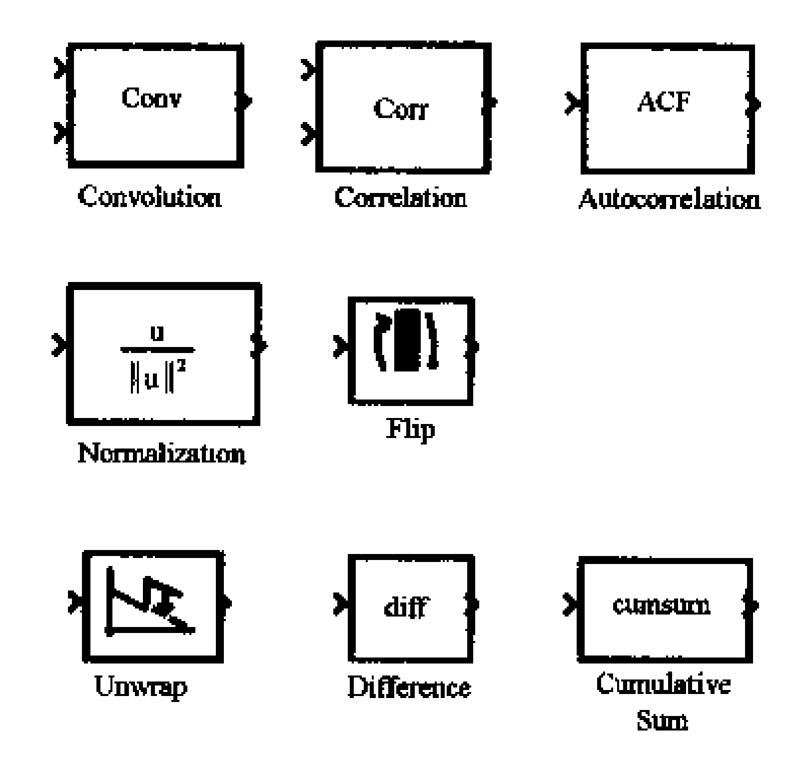


图 6.27　向量函数模块库中的模块

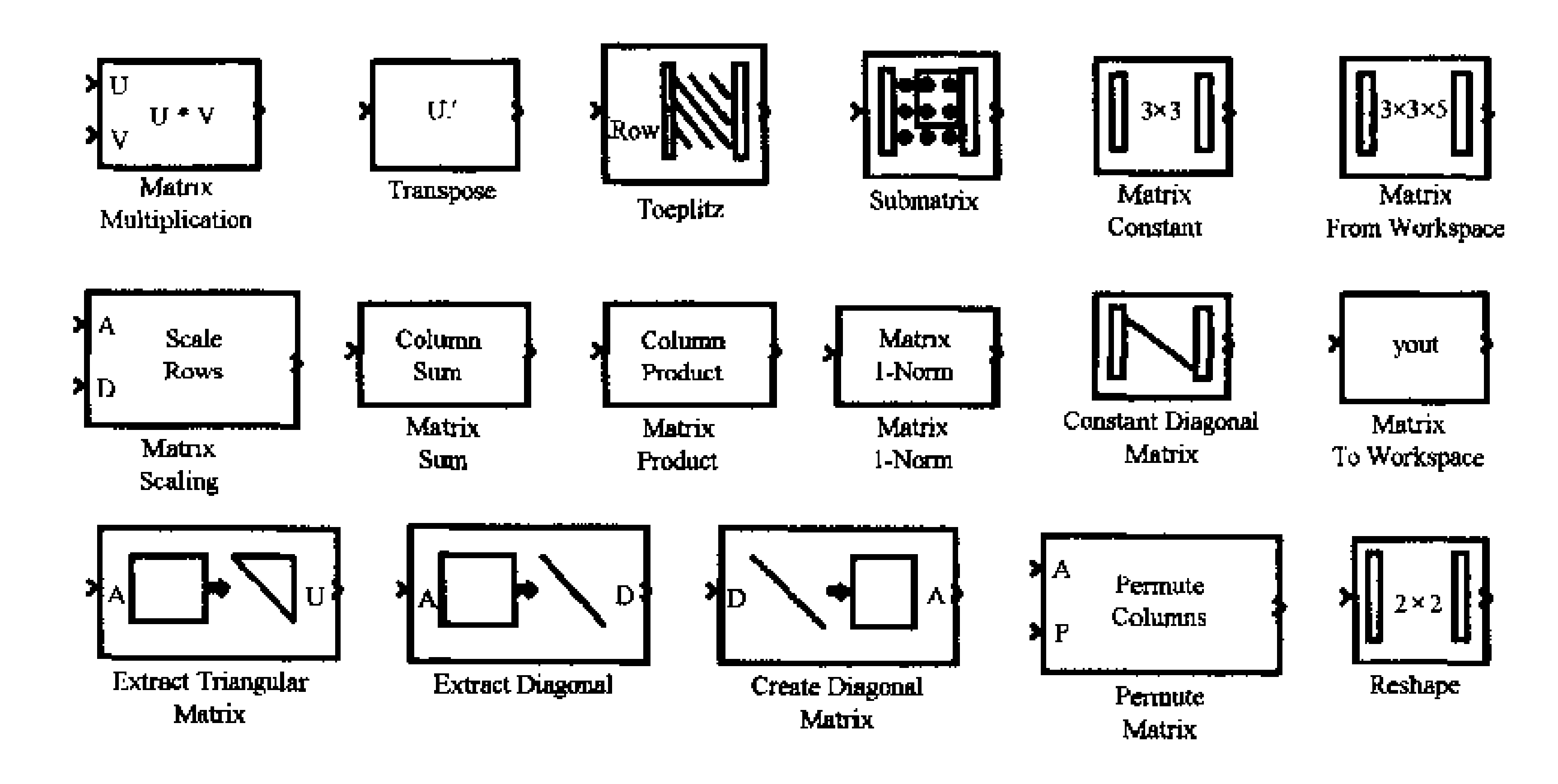


图 6.28　矩阵函数模块库中的模块

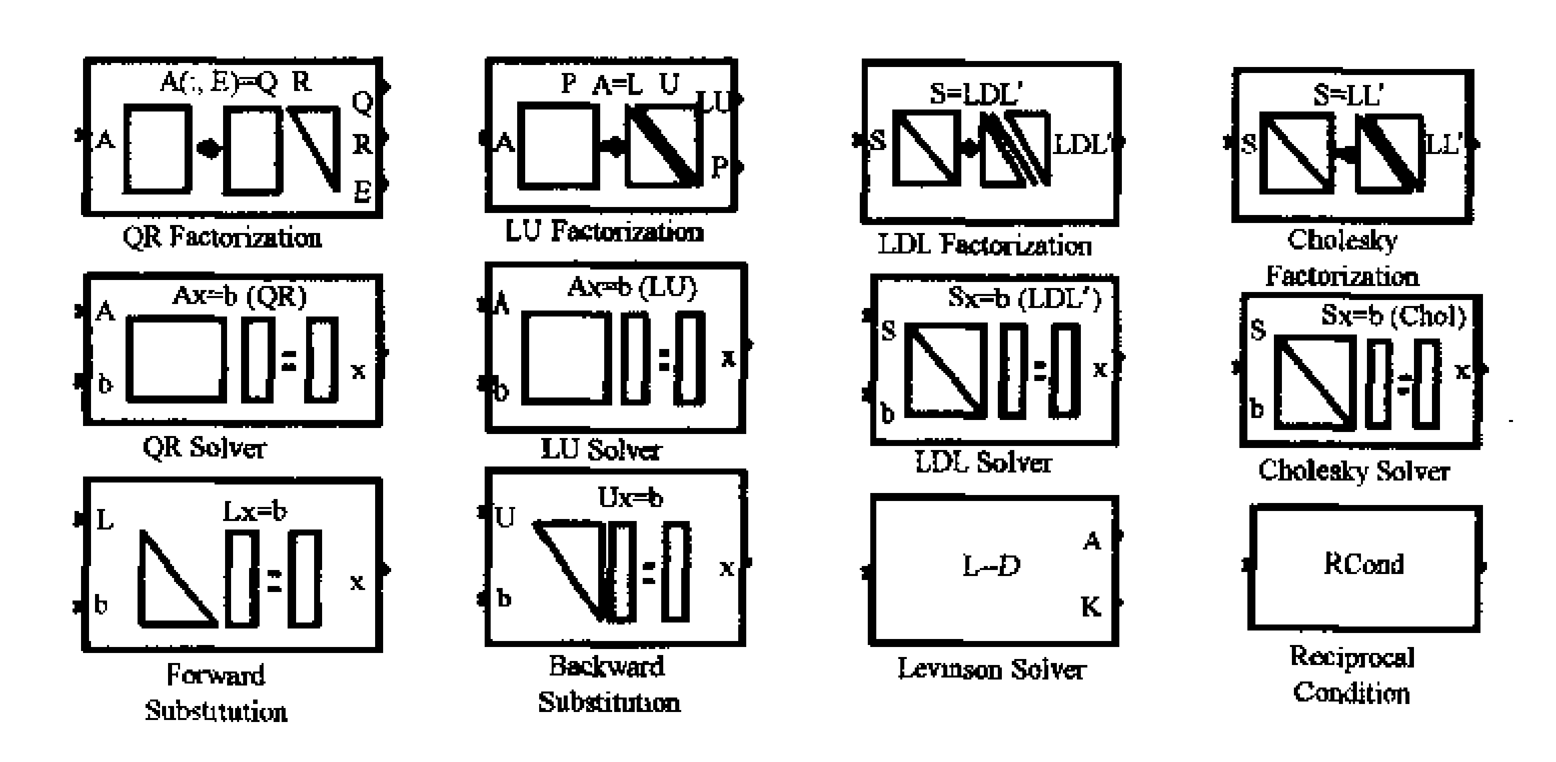


图 6.29　线性代数模块库中的模块

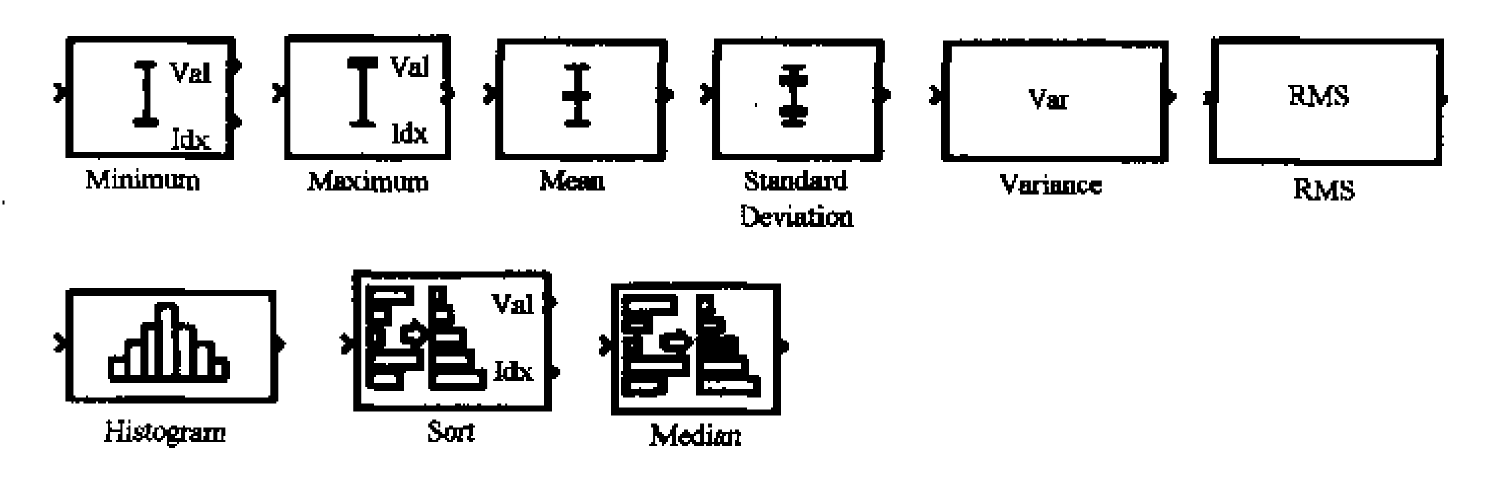


图 6.30　统计模块库中的模块

6.7 Fixed-Point Blockset(定点模块集)

定点模块集中的模块如图 6.31 所示. 包含的模块有:Filters and Systems Examples (滤波器与系统例子库),FixPt Constant(定点常数模块),FixPt Conversion(定点转换模块),FixPt Conversion Inherited(定点遗传转换模块),FixPt FIR(定点 FIR 模块),FixPt GUI(定点 GUI 模块),FixPt Gain(定点增益模块),FixPt Gateway In(定点门输入模块),FixPt Gateway Out(定点门输出模块),FixPt Logical Operator(定点逻辑运算模块),FixPt Look-Up Table(定点查找表模块),FixPt Look-Up Table 2-D(二维定点查找表模块),Fix-

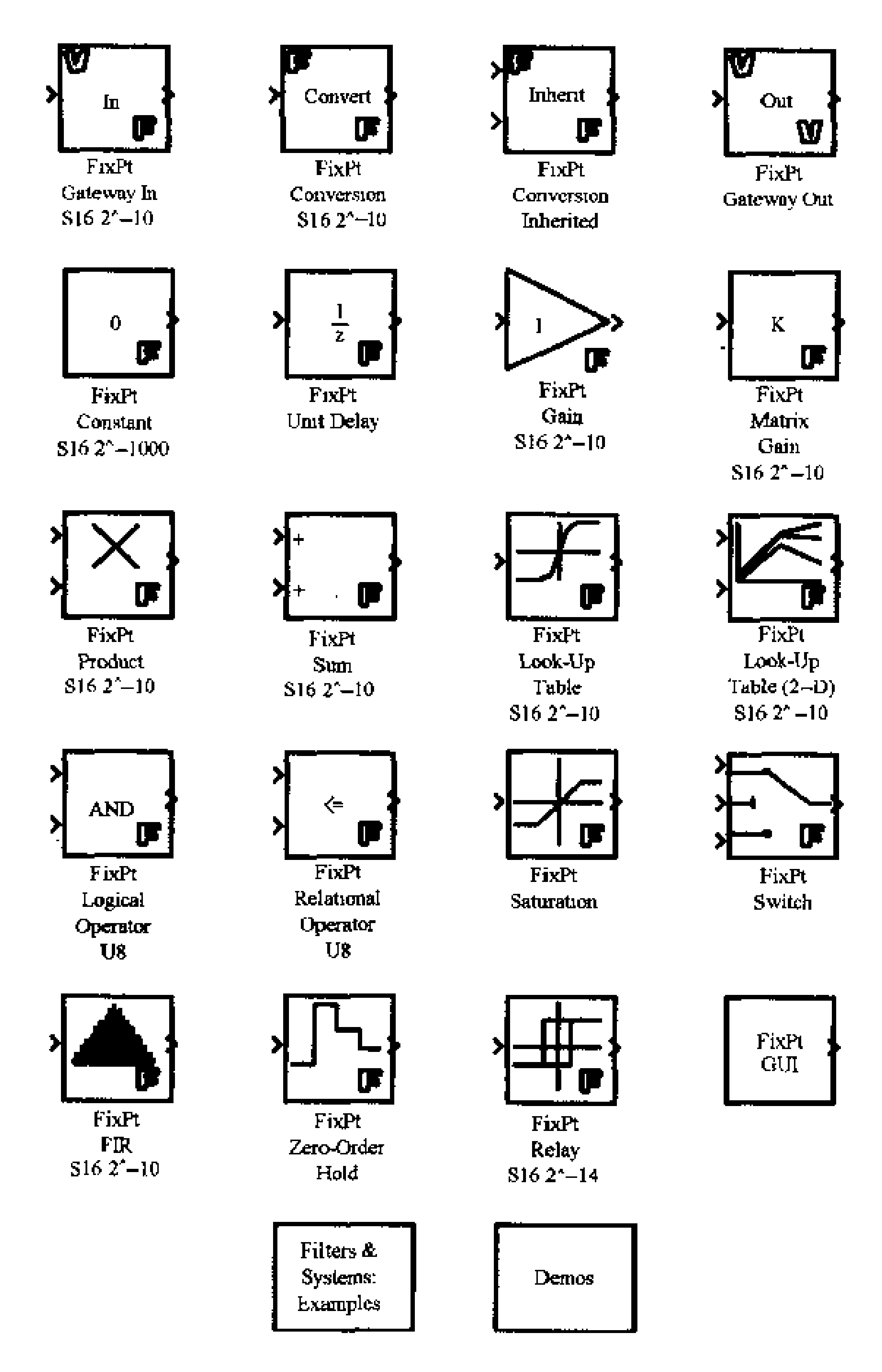


图 6.31 定点模块集中的模块

Pt Matrix Gain(定点矩阵增益模块),FixPt Product(定点乘积模块),FixPt Relation Operator(定点关系运算模块),FixPt Relay(定点延迟模块),FixPt Saturation(定点饱和模块),FixPt Sum(定点和模块),FixPt Switch(定点开关模块),FixPt Unit Delay(定点单位延迟模块),FixPt Zero-Order Hold(定点零阶保持模块).

6.8　Fuzzy Logic Toolbox(模糊逻辑工具箱)

模糊逻辑工具箱包含的模块有:Fuzzy Logic Controller(模糊逻辑控制器),Fuzzy Logic Controller with Ruleviewer(带有规则浏览器的模糊逻辑控制器),如图 6.32 所示.

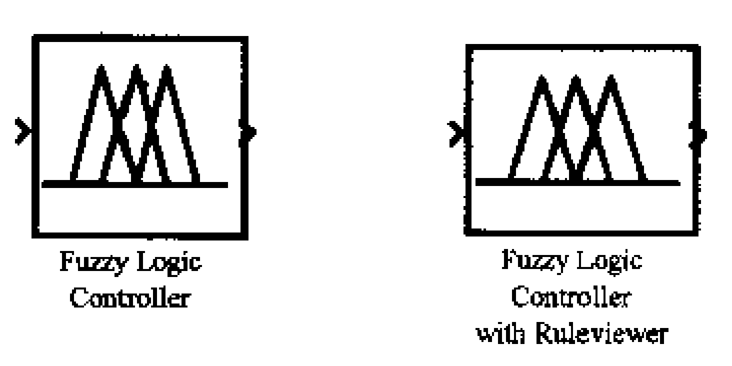


图 6.32　模糊逻辑工具箱中的模块

6.9　NCD Blockset(NCD 模块集)

NCD(Nonlinear Control Design,非线性控制设计)模块集包含的模块库有:RMS Blocks(RMS 模块库:其中含有连续、离散 RMS 模块,及演示模块),NCD Outport(NCD 输出端口模块),NCD Demos(NCD 演示模块库:其中含有四个演示模块和三个指南模块),如图 6.33 所示.

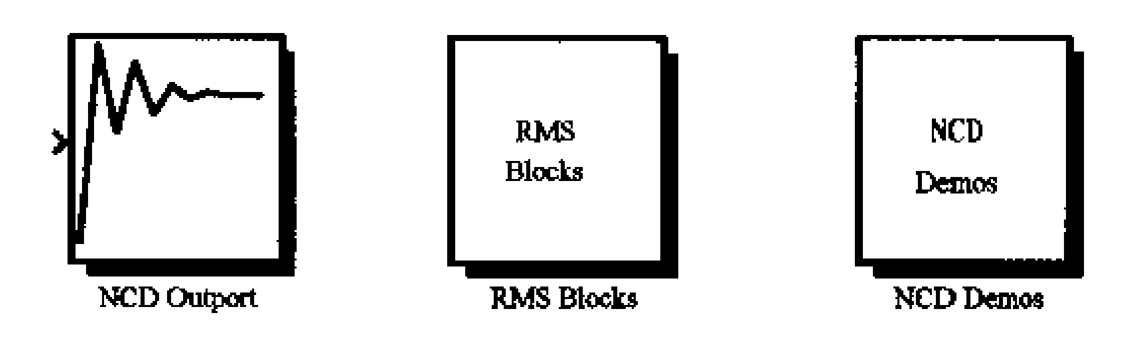


图 6.33　NCD 模块集中的模块库

6.10　Neural Network Blockset(神经网络模块集)

包含的模块库有:Net Input Functions(网络输入函数库),Transfer Functions(传递函数库),Weight Functions(权函数库).如图 6.34 所示.各库中的模块如图 6.35 至图 6.37 所示.

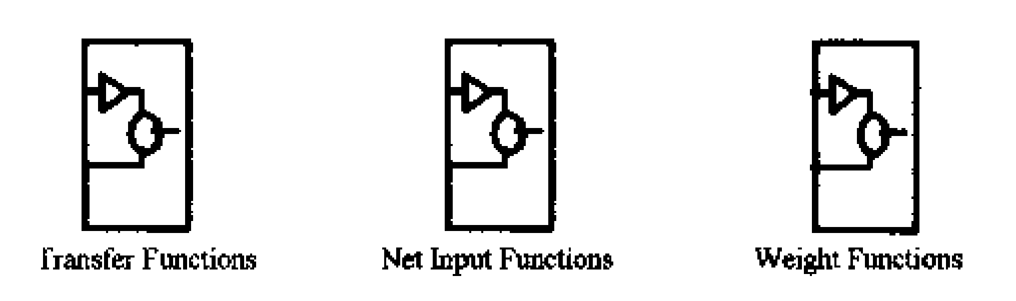


图 6.34　神经网络模块集中的模块库

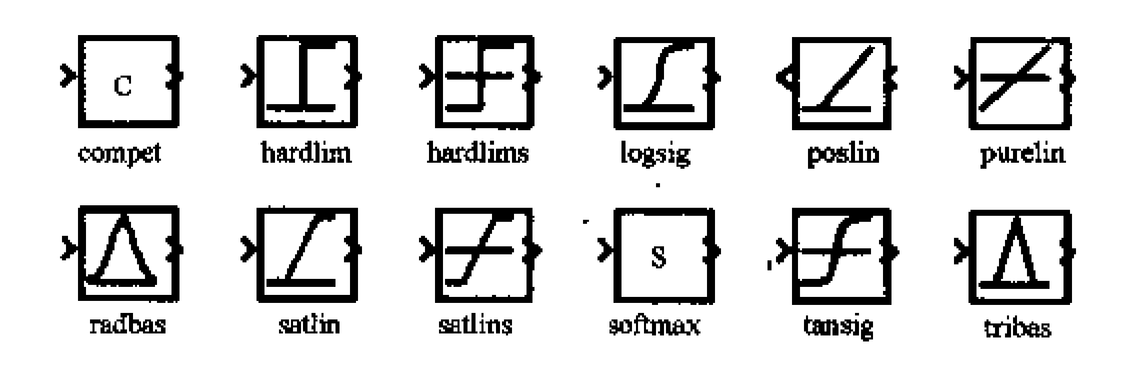


图 6.35　神经网络模块集中传递函数模块库中的模块

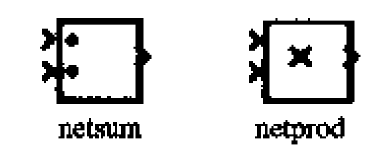


图 6.36　神经网络模块集中网络输入函数模块库中的模块

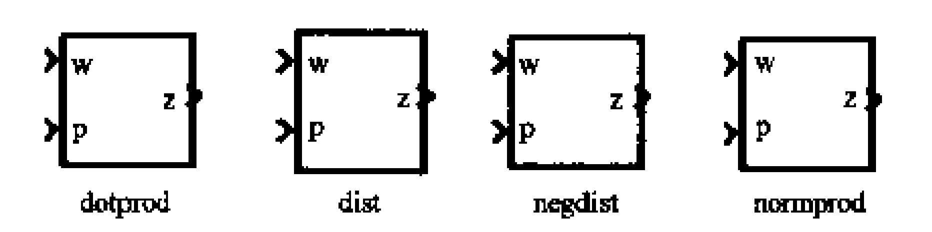


图 6.37　神经网络模块集中权函数模块库中的模块

6.11　MPC Blockset(MPC 模块集)

包含的模块有:nlcmpc 模块和 nlmpcsim 模块.

6.12　Power System Blockset(电源系统模块集)

电源系统模块集包含的模块库有:Connectiors(连接器库),Electrical Sources(电源库),Elements(元件库),Extra Libray(附加库),Machines(电机库),Measurements(仪表库),Power Electronics(电源电子元件库),powergui(电源图形用户界面模块).如图 6.38 所示.

1) 电源模块库中的模块如图 6.39 所示.有:直流电压源、交流电压源、交流电流源、可控电压源、可控电流源五个模块.

2) 元件模块库中的模块如图 6.40 所示.有:串联 RLC 支路、串联 RLC 负载、并联

RLC 支路、并联 RLC 负载、线性变压器、饱和变压器、互感器、电涌放电器、分布参数线路、断路器、π 截面导线等十一个模块.

3）电源电子元件模块库中的模块如图 6.41 所示.有：理想开关、金属氧化物半导体场效应晶体管、门电路、二级管、可控硅等六个模块.

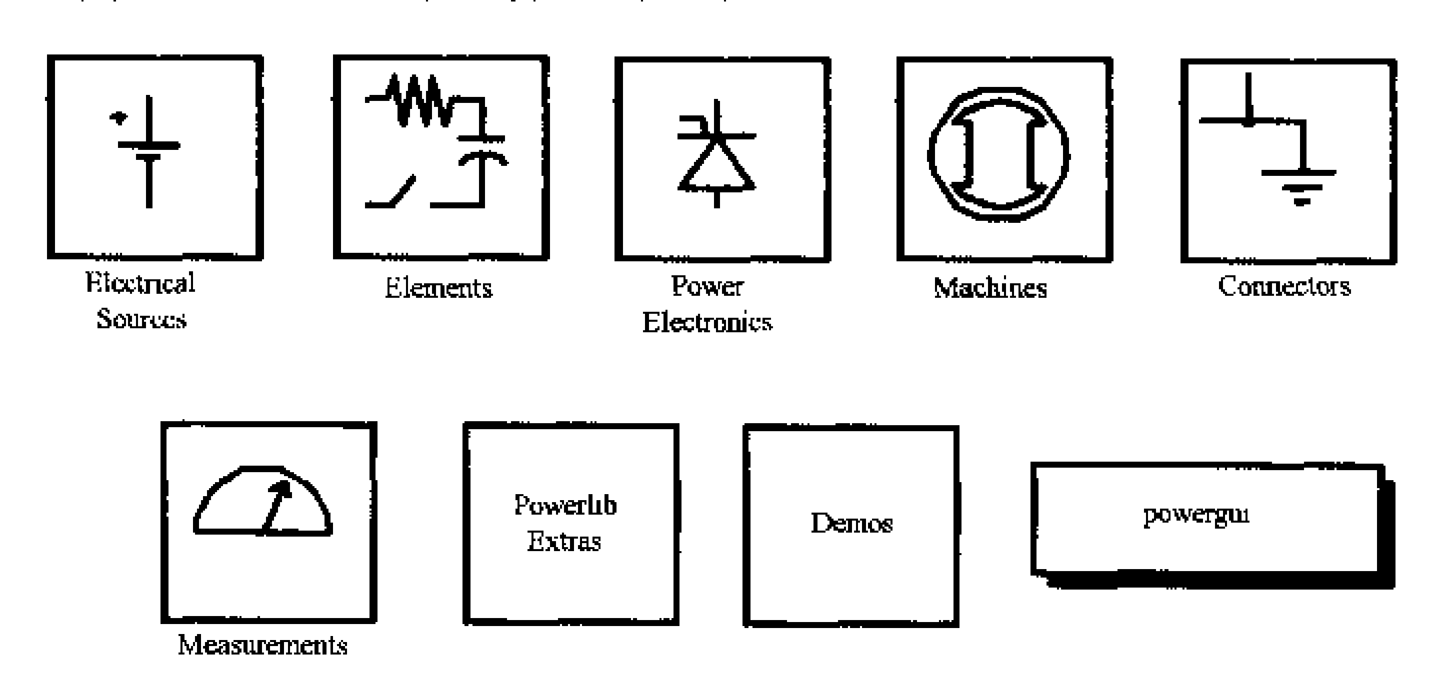


图 6.38　电源系统模块集中的模块库

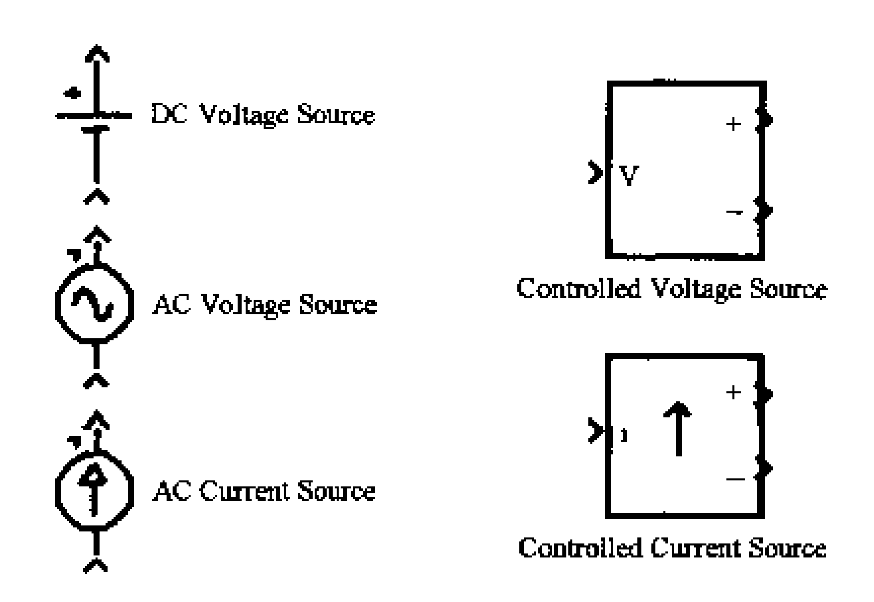


图 6.39　电源系统模块集电源模块库中的模块

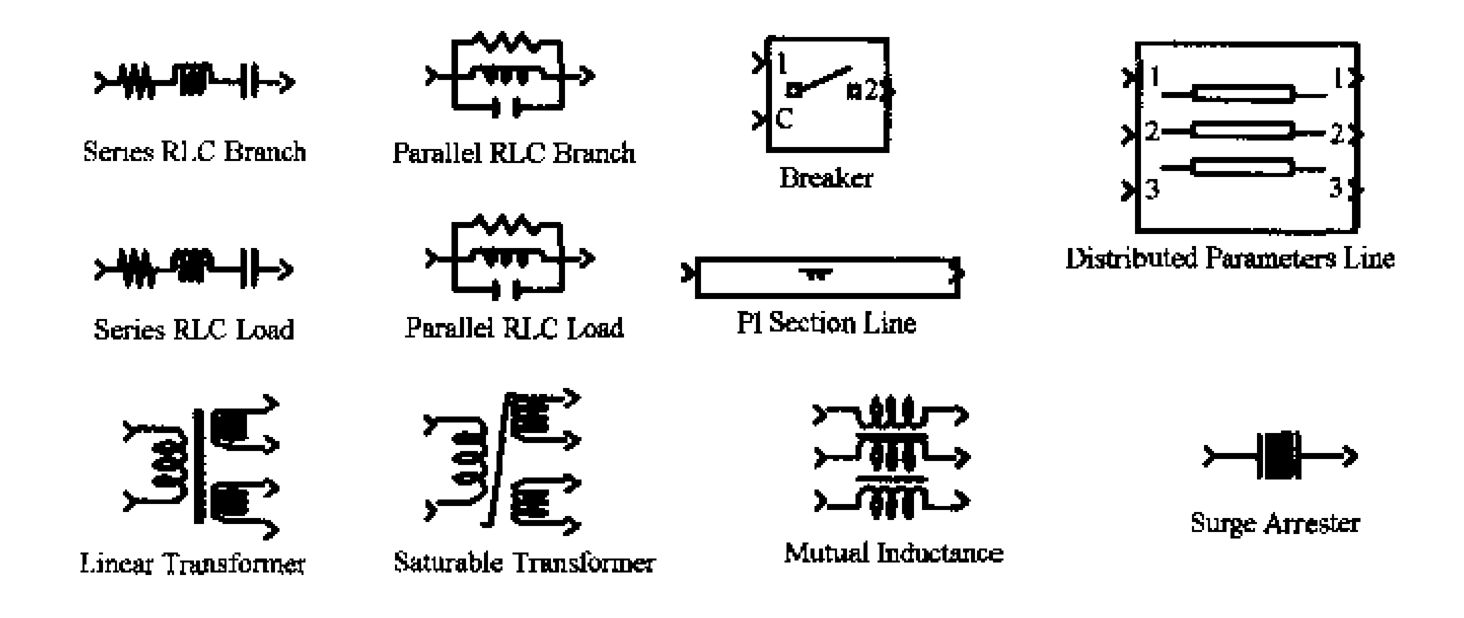


图 6.40　电源系统模块集元件模块库中的模块

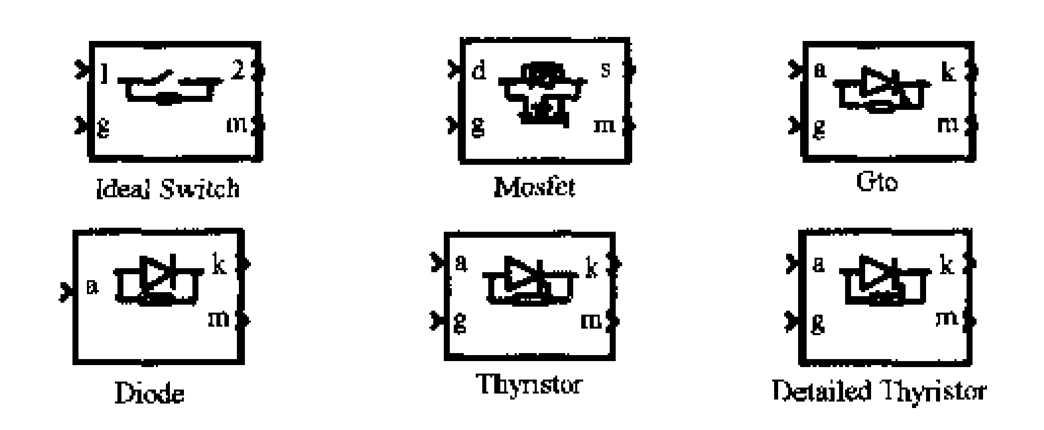


图 6.41 电源系统模块集电子元件模块库中的模块

4）仪表模块库中的模块如图 6.42 所示. 有：电压测量和电流测量两个模块.

5）连接器模块库中的模块如图 6.43 所示. 包含有：接地（输入、输出）两个模块，局部接地（输入、输出）两个模块，T 型和 L 型连接器两个模块，多进多出连接器（水平、垂直）两个模块，多进多出薄连接器（水平、垂直）两个模块.

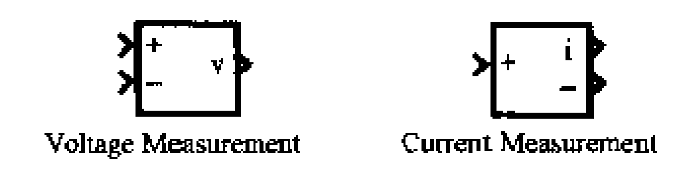


图 6.42 电源系统模块集仪表模块库中的模块

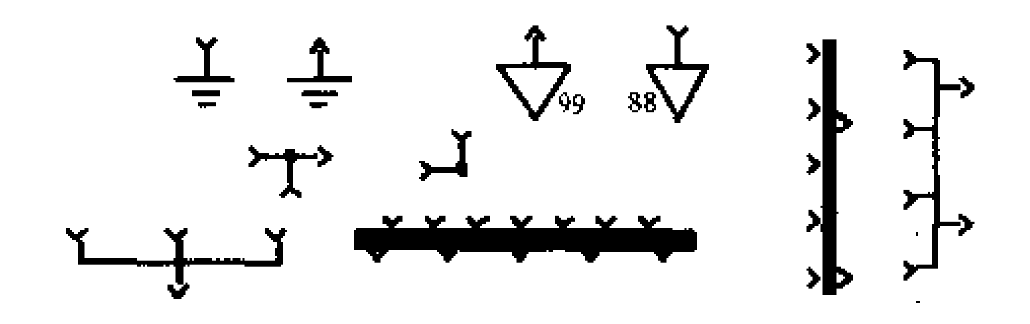


图 6.43 电源系统模块集连接器模块库中的模块

6）电机模块库中的模块如图 6.44 所示. 包括简单同步电机三个模块，恒磁同步电机两个模块，异步电机三个模块，涡轮与调节器两个模块，同步电机四个模块.

7）电源库附加模块库又包括：测量模块库，三相模块库，控制模块集和附加电机模块库，如图 6.45 所示.

测量模块库，有三个模块，如图 6.46 所示.

三相模块库，有十五个模块，如图 6.47 所示.

控制模块集，有定时器和同步 6 脉冲发生器两个模块，如图 6.48 所示.

附加电机库，有一个直流电机模块，如图 6.49 所示.

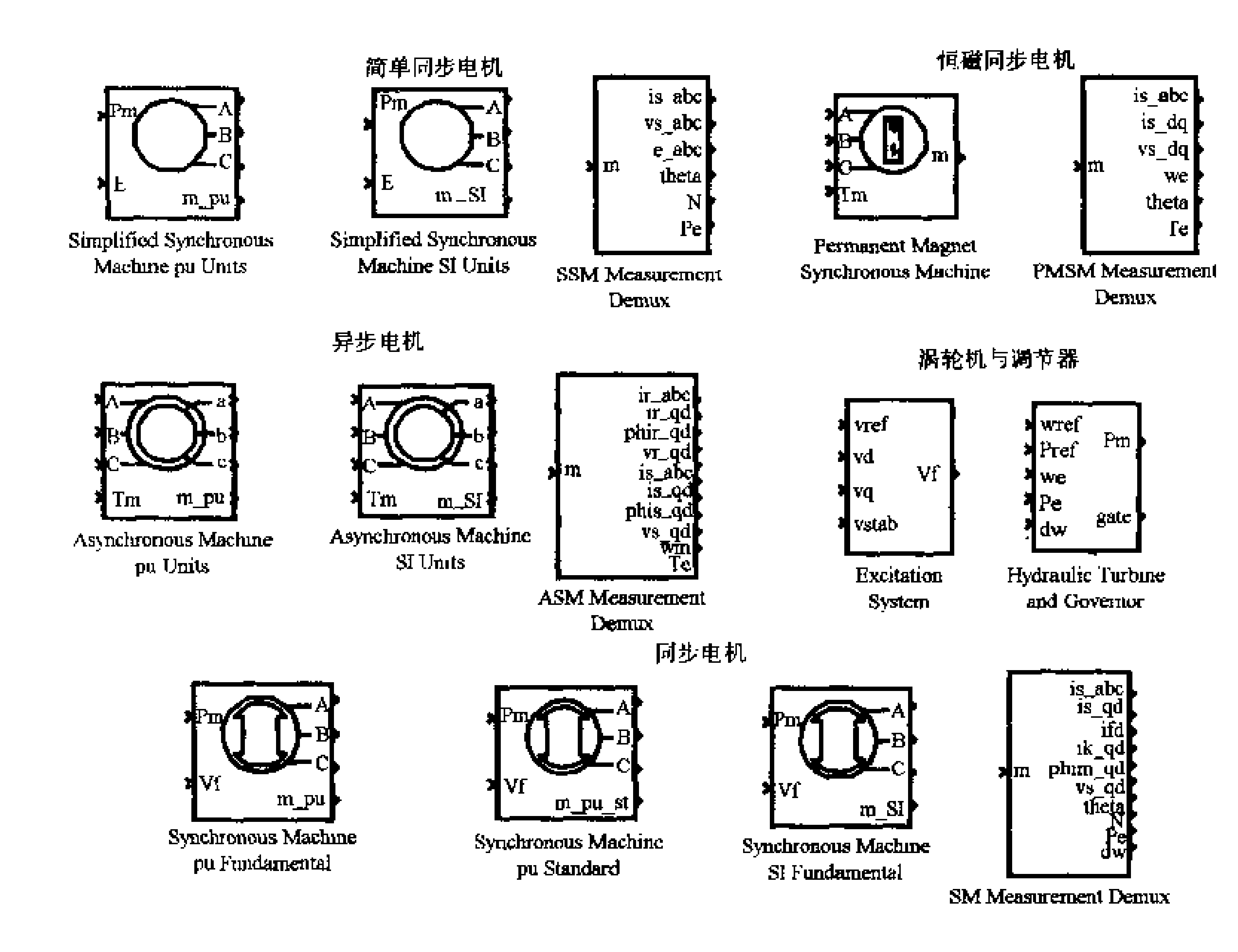


图 6.44　电源系统模块集电机模块库中的模块

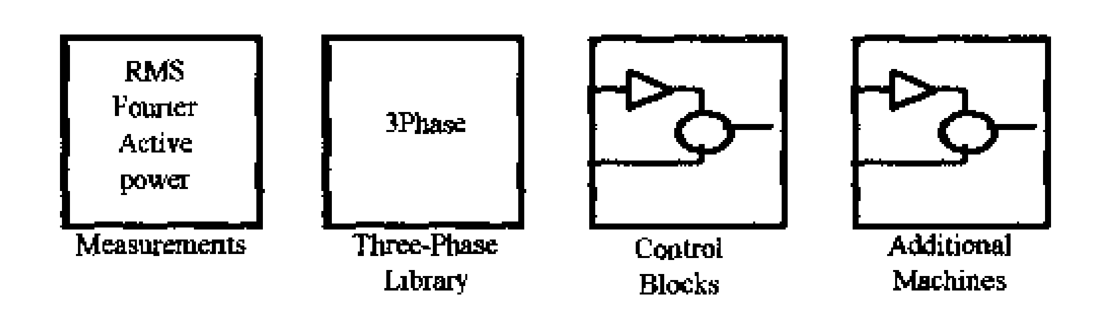


图 6.45　附加模块库中的库

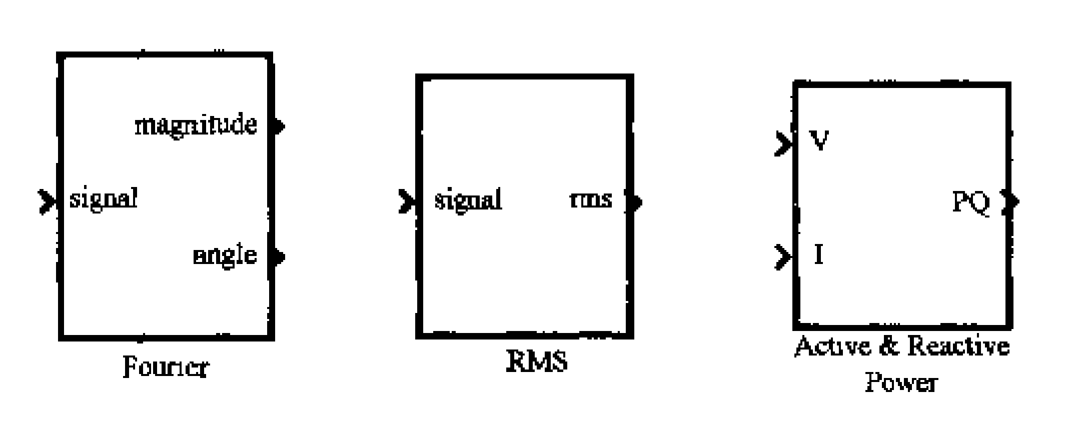


图 6.46　测量库中的模块

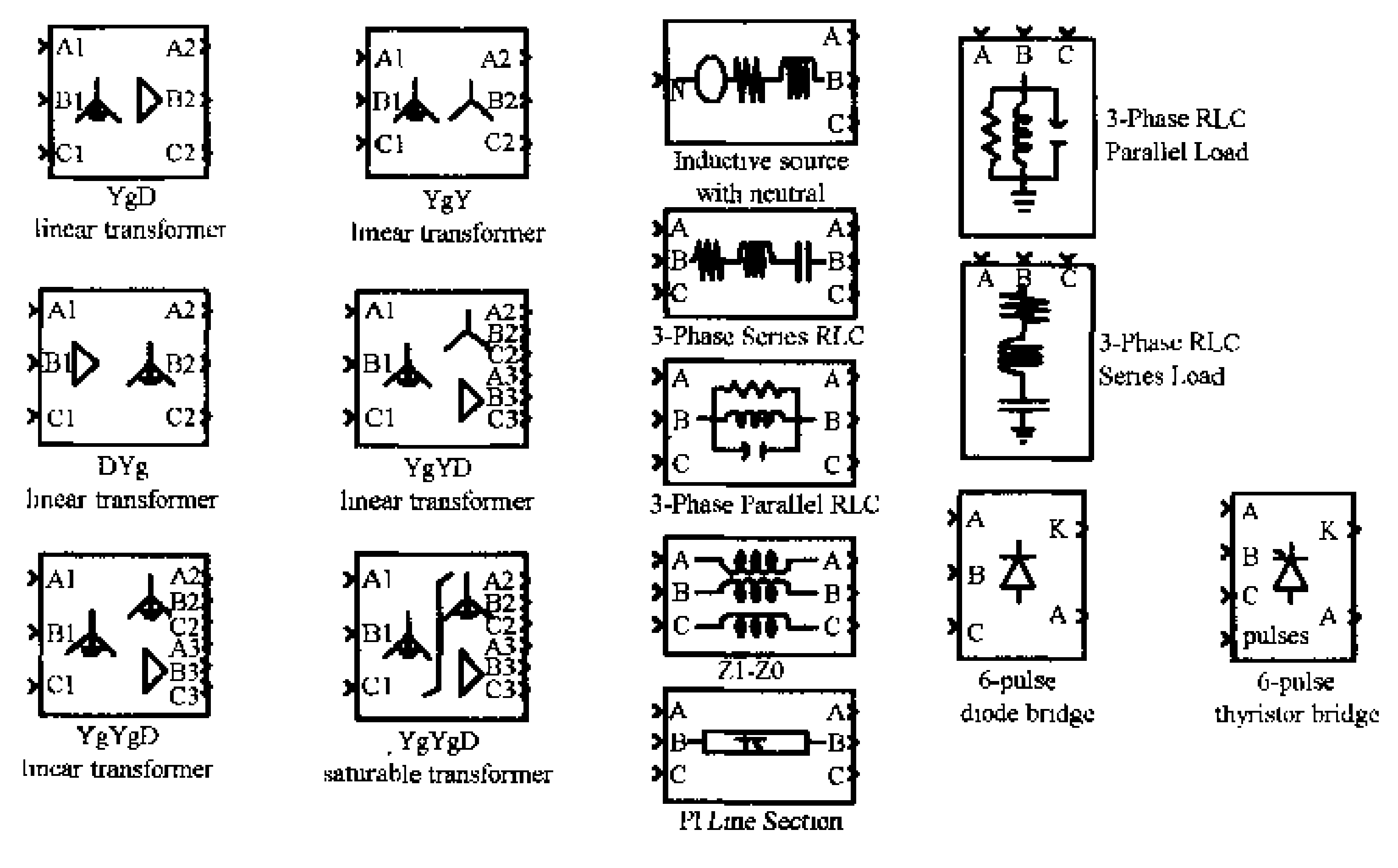


图 6.47　三相库中的模块

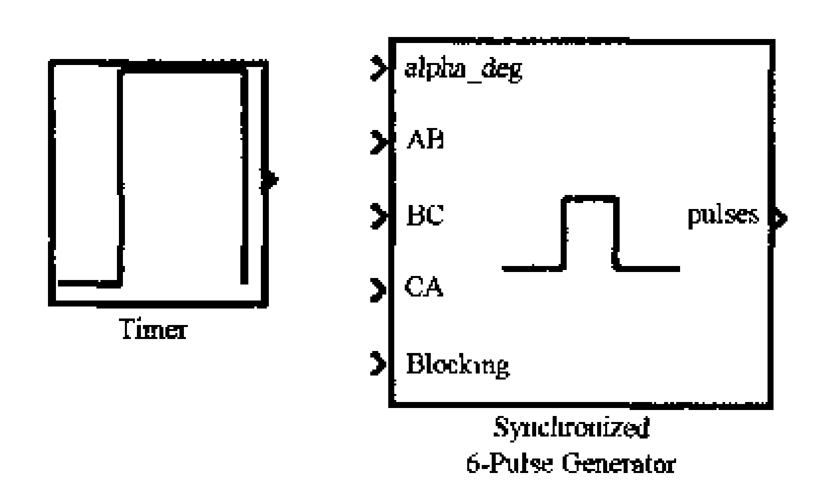


图 6.48　控制模块集中的模块

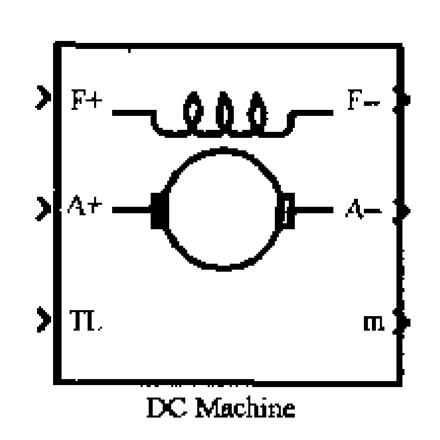


图 6.49　附加电机库中的模块

6.13　Real-Time Windows Target(实时窗口目标库)

实时窗口目标库包含的模块有:Adapter(适配器模块),RT In(实时输入单元模块),RT Out(实时输出单元模块),以及演示模块库.演示库中有:实时 VDP(Van Der Pol)演示模块、实时信号发生器模块、实时滤波器模块和实时控制器模块.

6.14　Real-Time Workshop(实时工作空间库)

实时工作空间库包含的模块库有:Custom Code(自定义代码库),DOS Device Drivers (DOS 设备驱动器库),Interrupt Templates(中断模板库),S-Function Target(S 函数对象),VxWorks Support(Vx 支持库).如图 6.50 所示.每个模块库中又可能包含几个子库.

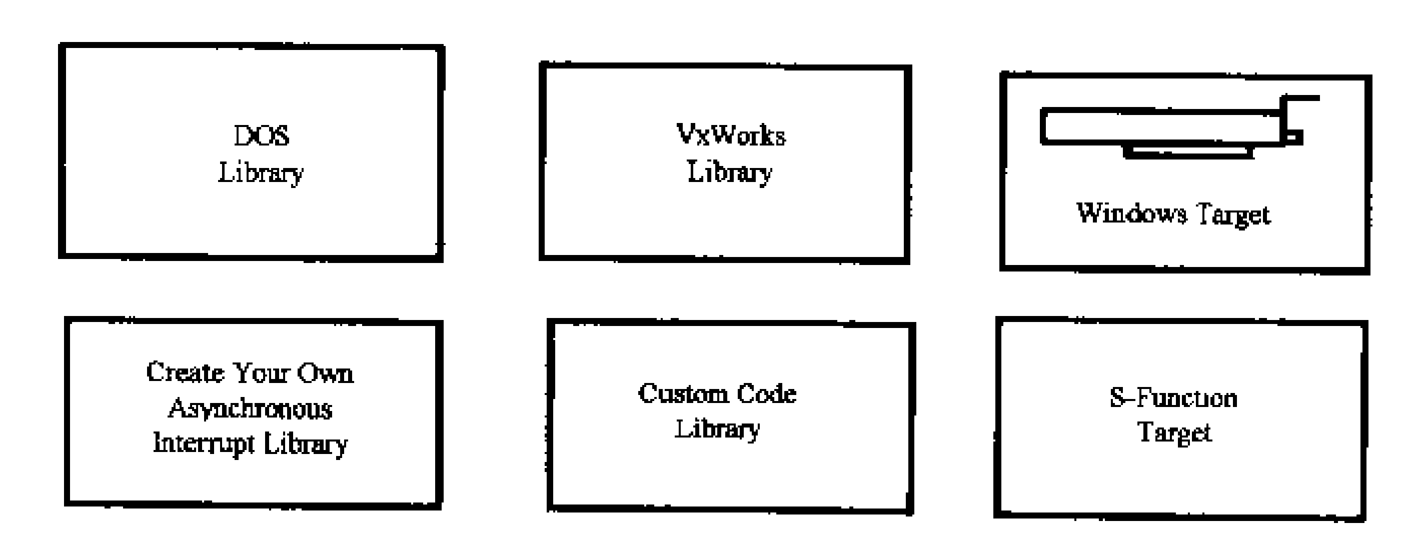


图 6.50　实时工作空间库中的模块库

6.15　Stateflow(状态流程库)

状态流程库包含有状态流程 Chart(图)模块和例子库.状态流程图窗口如图 6.51 所示.

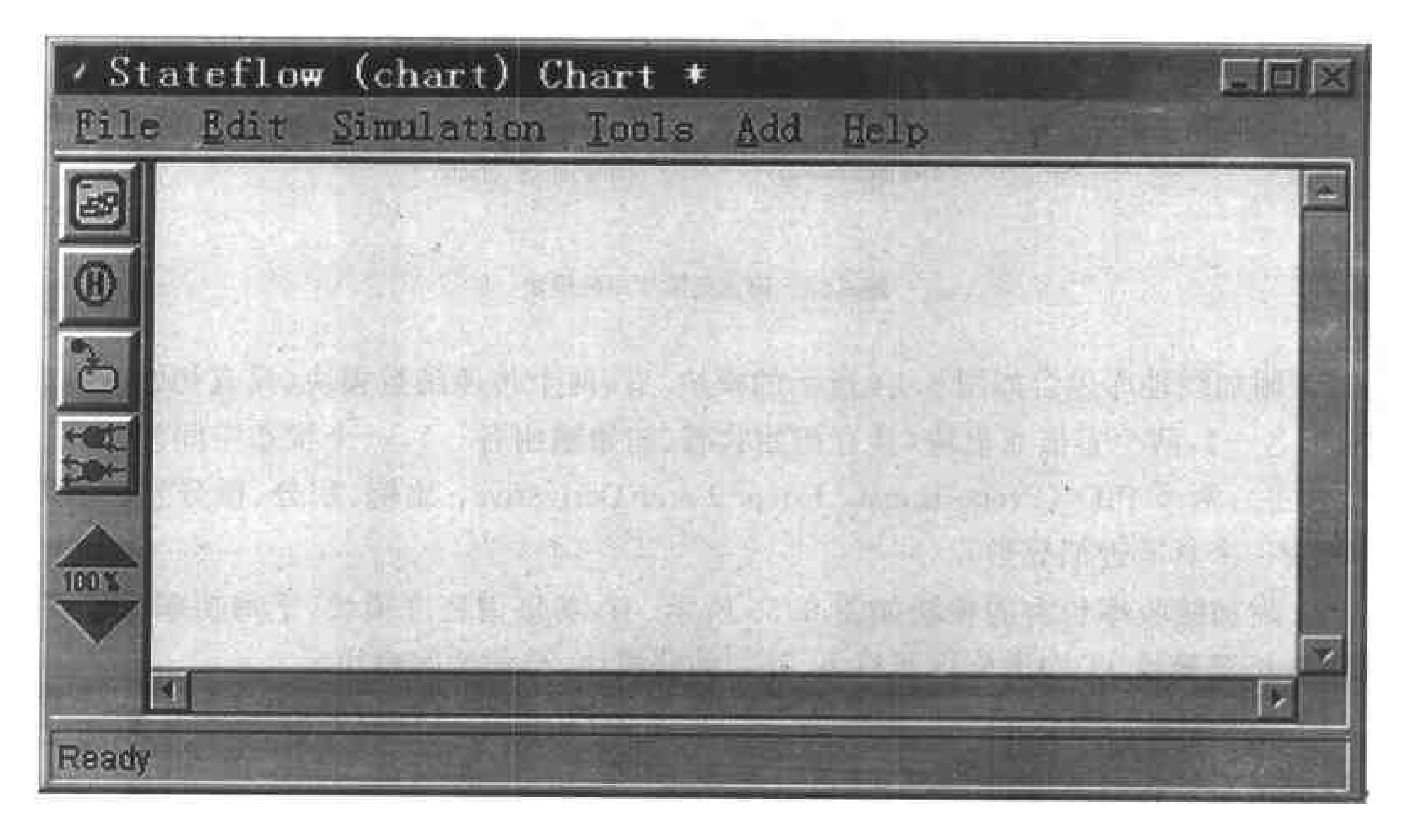


图 6.51　状态流程图窗口

6.16 Simulink Extras(Simulink 附加库)

Simulink 附加库包含的模块库有:Additional Discrete(附加离散库),Additional Linear(附加线性库),Additional Sinks(附加接收库),Flip Flops(触发器库),Linearization(线性化库),Transformations(转换库),如图 6.52 所示.

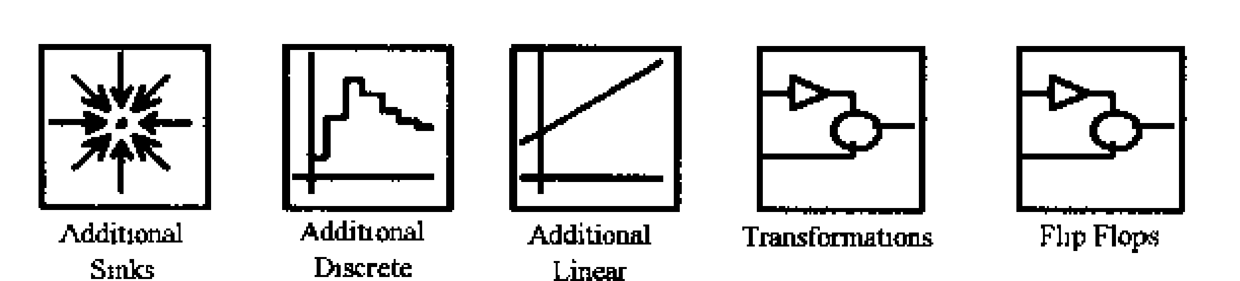


图 6.52 Simulink 附加库中的模块库

1) 附加离散库包含的模块如图 6.53 所示.有:两个离散传递函数模块(具有初始状态、初始输出各一),两个离散零极点模块(具有初始状态、初始输出各一).

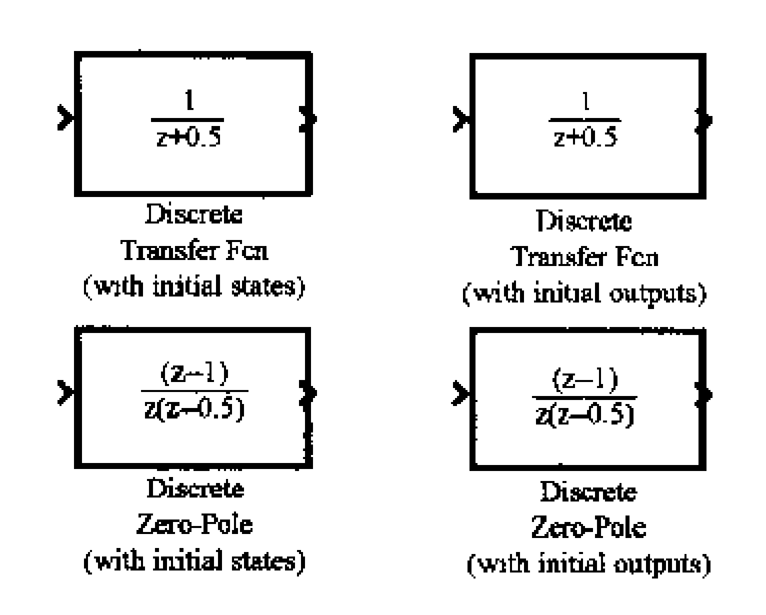


图 6.53 附加离散库中的模块

2) 附加线性库包含如图 6.54 所示的模块.有:两个传递函数模块(具有初始状态、初始输出各一),两个零极点模块(具有初始状态、初始输出各一),一个状态空间模块(具有初始输出),两个 PID (Proportional, Integral and Derivative, 比例、积分、微分)控制器模块(其中一个具有近似导数).

3) 附加接收库包含的模块如图 6.55 所示.有:功率谱密度模块,平均功率谱密度模块,谱分析器模块,平均谱分析器模块,互相关器模块,自相关器模块.

4) 触发器库包含的模块如图 6.56 所示.有:时钟模块,D 锁存器模块,S-R 触发器模块,D 触发器模块,J-K 触发器模块(负沿触发).

5) 线性化库包含的模块如图 6.57 所示.包括线性化转换导数、转换传递延迟两个模块.

6) 转换库中包含的模块如图 6.58 所示.有:极坐标与笛卡儿坐标相互转换两个模

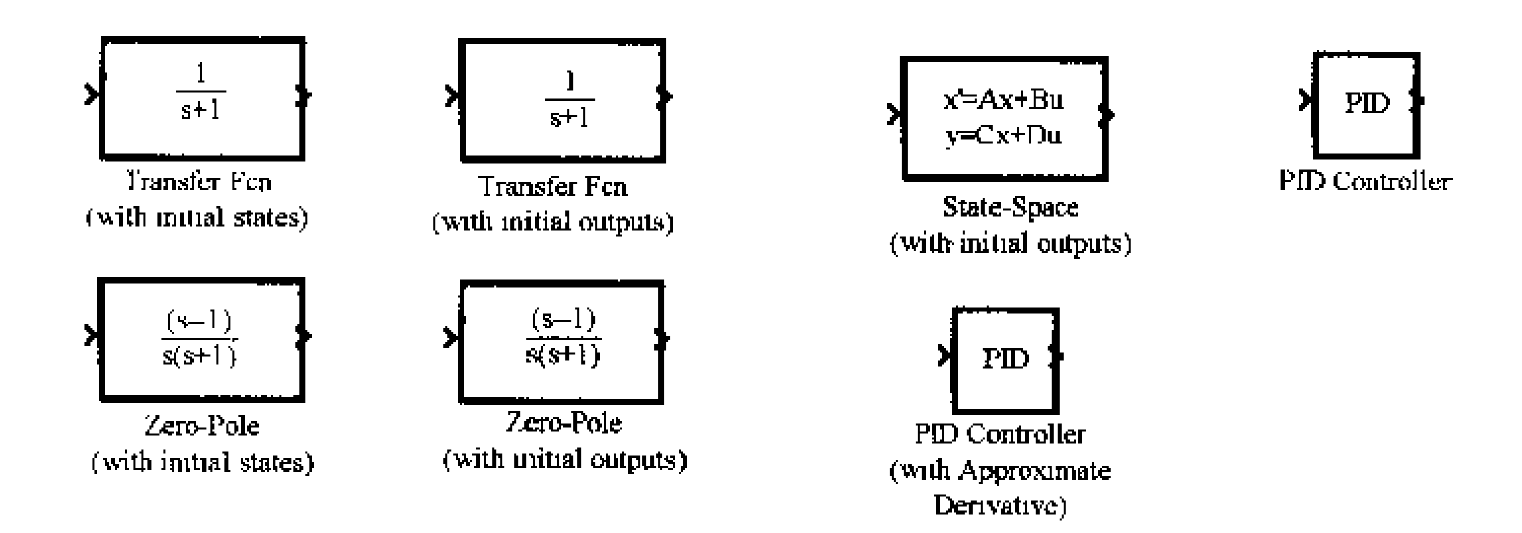


图 6.54 附加线性库中的模块

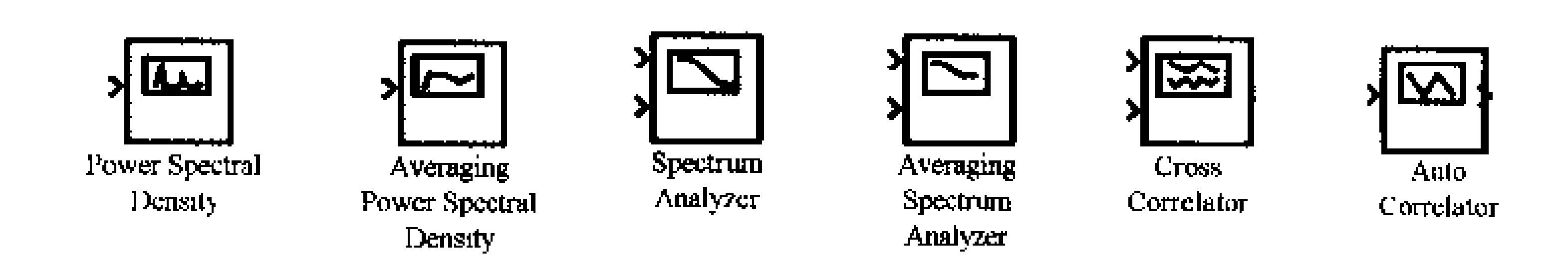


图 6.55 附加接收库中的模块

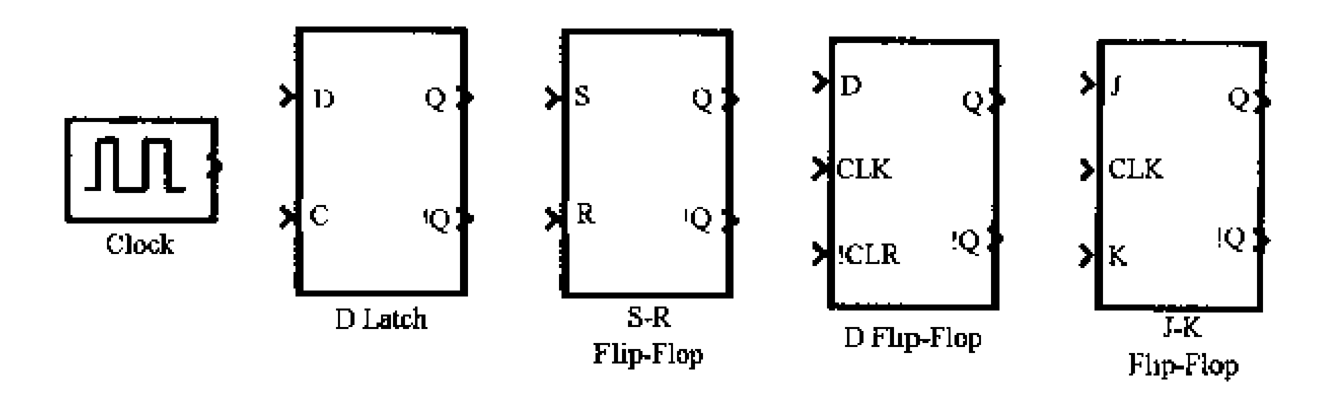


图 6.56 触发器库中的模块

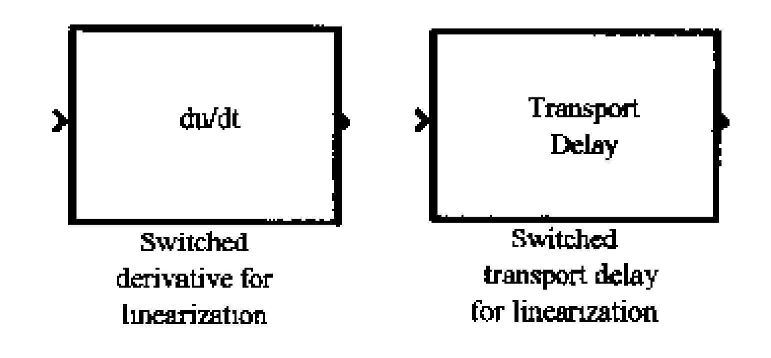


图 6.57 线性化库中的模块

块，华氏温度与摄氏温度相互转换两个模块，球坐标与笛卡儿坐标相互转换两个模块，度与弧度相互转换两个模块.

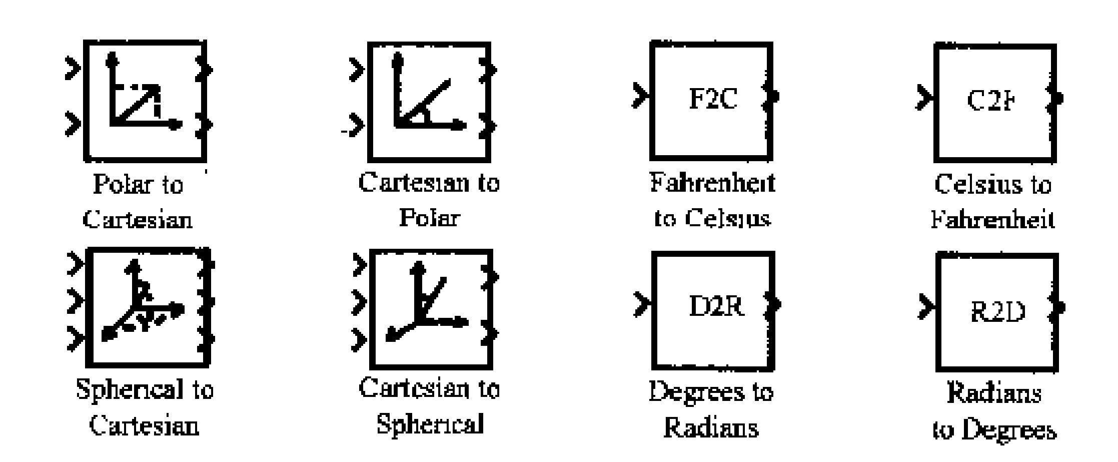


图 6.58 转换库中的模块

6.17 System ID Blocks(系统辨识模块集)

系统辨识模块集包含的模块有：AutoRegressive model estimator(自回归模型估计模块，简称 AR 模块)，AutoRegressive with eXternal input model estimator(带有外部输入的自回归模型估计模块，简称 ARX 模块)，AutoRegressive Moving Average with external input model estimator(带有外部输入的移动平均自回归模型估计模块，简称 ARMAX 模

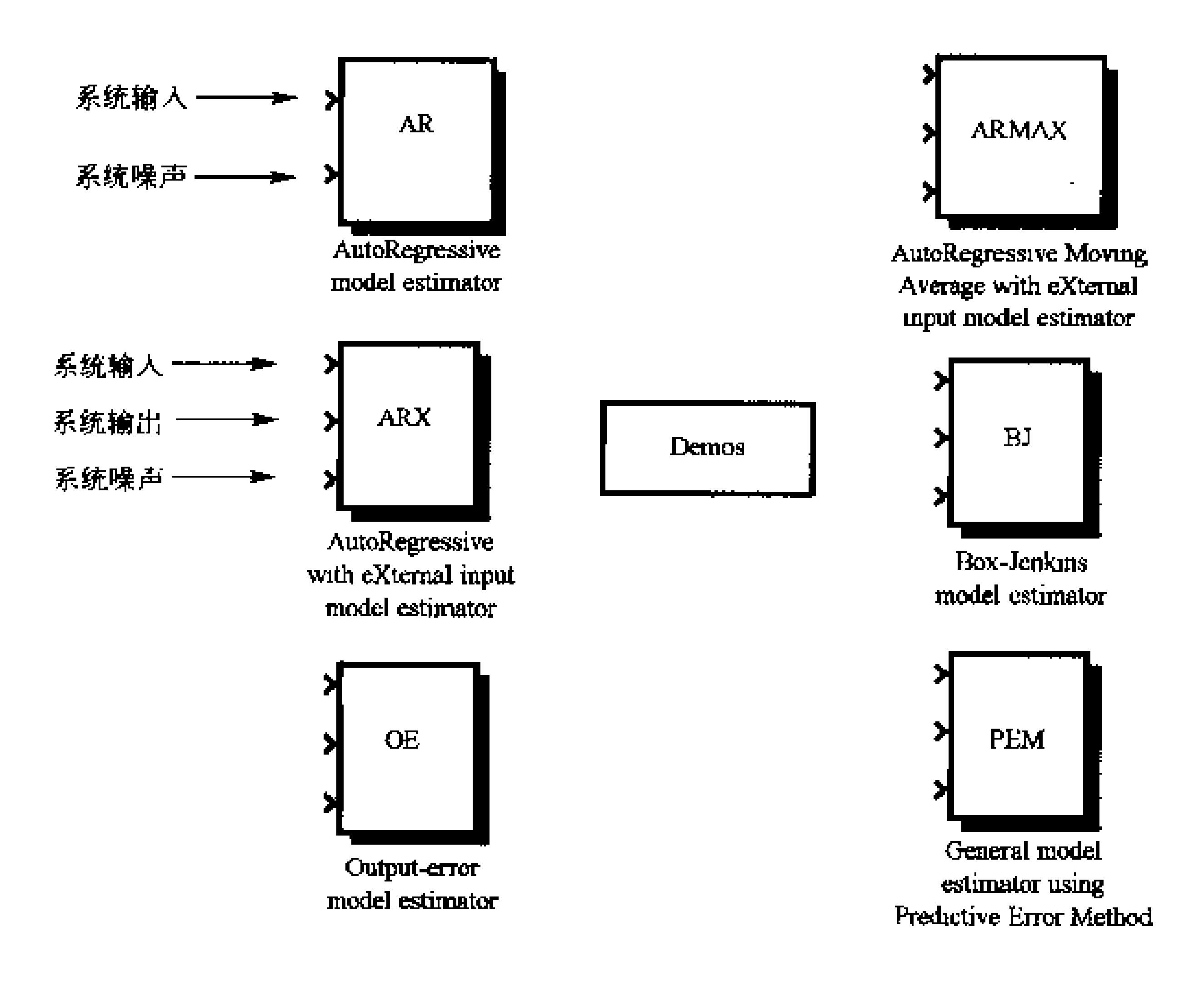


图 6.59 系统辨识模块集中的模块

块),Box-Jenkins model estimator(Box-Jenkins 模型估计模块,简称 BJ 模块),General model estimator using Predictive Error Method(使用误差推测方法的通用模型估计模块,简称 PEM 模块),Output-error model estimator(输出误差估计模块,简称 OE 模块),以及演示模块,如图 6.59 所示.

第七章　Simulink 模块库与模块

在 6.2 节中已经简要介绍了 Simulink 模块库及各库中所包含的模块. 在这一章中将详细介绍这些模块的作用和用法.

7.1　Sources 库中的模块

Sources 库中包含的模块如表 7.1 所列,各个模块的图标如图 6.10 所示.

表 7.1　Sources 库中的模块

模　块　名	功　　能
Band Limited White Noise	给连续系统引入白噪声
Chirp Signal	产生一个频率递增的正弦波(线性调频信号)
Clock	显示并提供仿真时间
Constant	生成一个常量值
Digital Clock	生成有着给定采样间隔的仿真时间
Digital Pulse Generator	生成有着规则间隔的脉冲
From File	从文件读取数据
From Workspace	从工作空间中定义的矩阵中读取数据
Pulse Generator	生成有着规则间隔的脉冲
Ramp	生成一连续递增或递减的信号
Random Number	生成正态分布的随机数
Repeating Sequence	生成一重复的任意信号
Signal Generator	生成变化的波形
Sine Wave	生成正弦波
Step	生成一阶跃函数
Uniform Random Number	生成均匀分布的随机数

7.1.1　Band-Limited White Noise(限带白噪声)

(1) 模块功能

Band-Limited White Noise 模块给一个连续系统引入白噪声.

(2) 模块说明

Band-Limited White Noise 模块生成正态分布的随机数,它们适用于连续或者混合系统. 这一模块与 Random Number 模块最主要的差别是 Band-Limited White Noise 模块以一给定的采样率产生输出,该采样率与噪声的相关时间有关.

理论上,连续白噪声的相关时间为 0,功率谱密度图(PSD)是平坦的,协方差无限大.实际上,物理系统受到的干扰并不是白噪声,当干扰噪声的相关时间相对于系统的带宽来说非常小时,白噪声不失为一个有用的理论近似.

在 Simulink 中,可以用相关时间比系统的最短时间常数小得多的随机序列来模拟白噪声的效果. Band-Limited White Noise 模块产生这样的一个序列,其噪声的相关时间是模块的采样速率.要想精确地进行仿真,必须使用比系统最快的动态分量还要小得多的相关时间.按式(7.1)给定相关时间将得到比较好的结果.

$$t_c = \frac{2\pi}{100 f_{max}} \tag{7.1}$$

式中,f_{max}是系统的带宽,用弧度/秒表示.

(3)模块数据类型

该模块输出数据类型为双精度实数值.

(4)模块参数对话框

该模块的参数和对话框如图 7.1 所示.

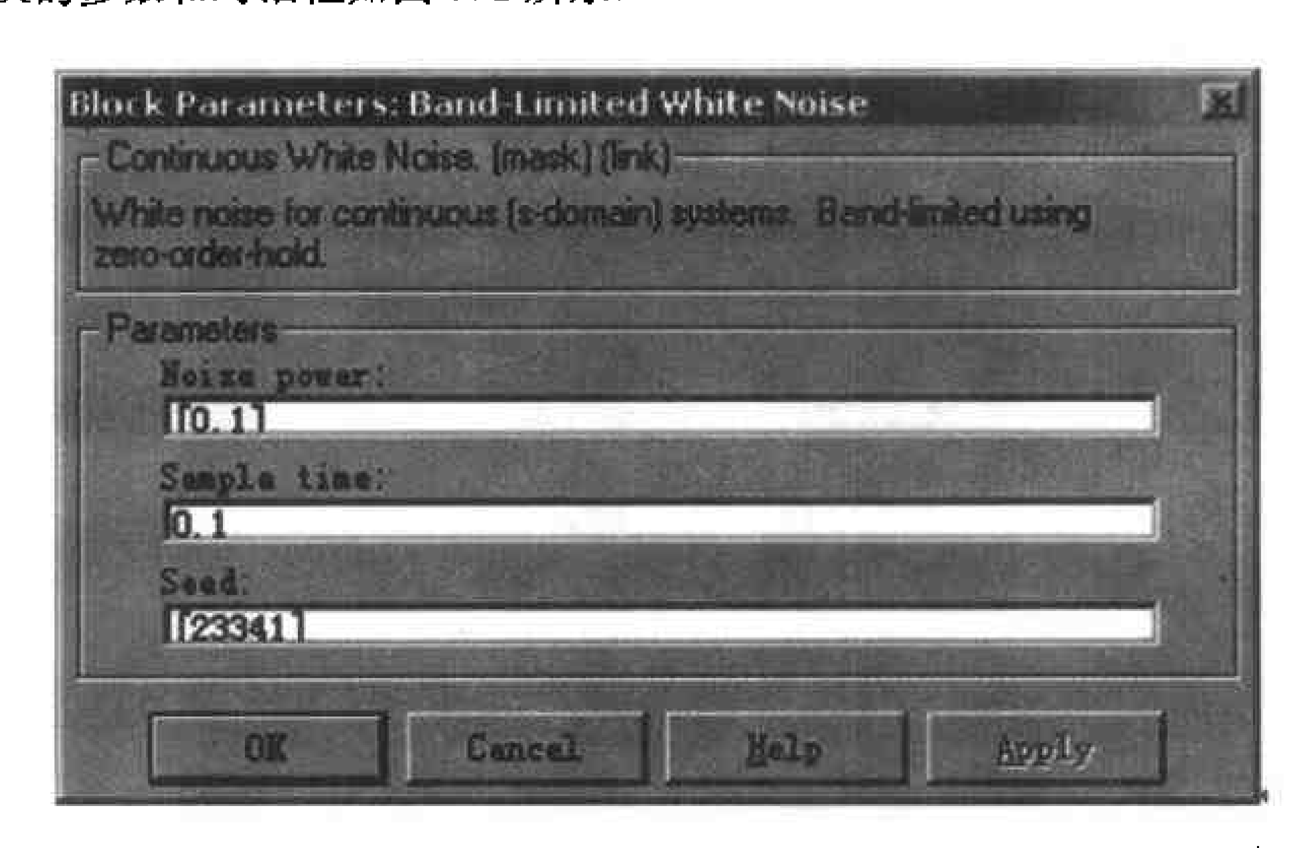


图 7.1　Band-Limited White Noise 模块参数对话框

噪声功率(Noise power),白噪声的 PSD 的高度.缺省值为:0.1.

采样时间(Sample time),噪声相关时间.缺省值为:0.1.

种子(Seed),发生随机的开始种子.缺省值为:23341.

(5) 模块特点

1) 离散采样时间;

2) 噪声功率参数、种子参数和输出有标量扩展;

3) 可向量化;

4) 没有过零区间.

7.1.2 Chirp Signal(扫频信号)

(1) 模块功能

产生一频率递增的正弦波信号.

(2) 模块说明

Chirp Signal 模块产生一频率随时间线性递增的正弦波信号. 可以用该模块进行非线性系统的频谱分析. 该模块的输出为标量或向量.

(3) 模块数据类型

该模块输出为双精度类型实数值信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.2 所示.

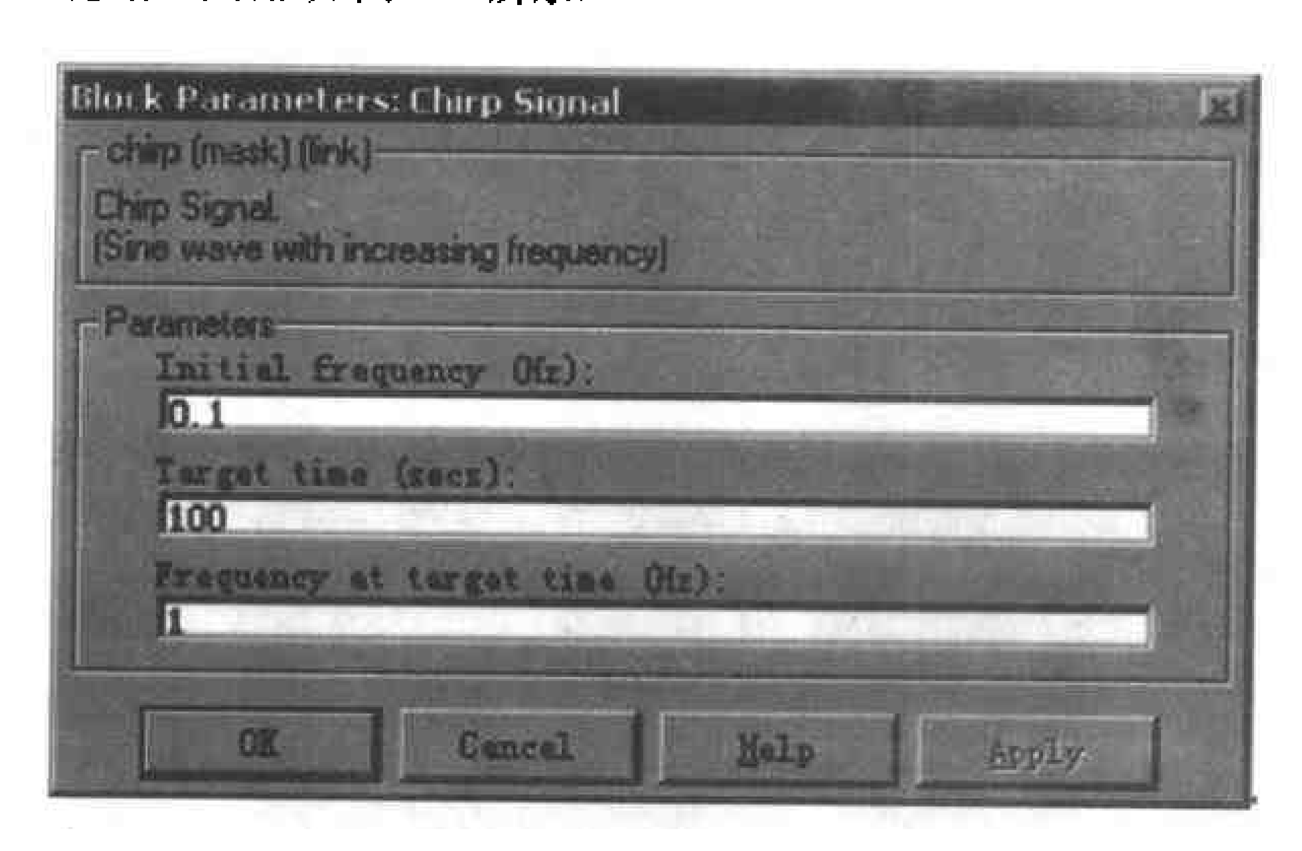


图 7.2 Chirp Signal 模块的参数对话框

1) 初始频率(Initial frequency), 信号的初始频率指定为标量或向量值. 缺省值为: 0.1Hz.

2) 目标时间(Target time), 就是频率达到目标频率参数的时间值, 为一标量或向量值. 在该时间后, 频率以相同的速率连续改变. 缺省值为 100 秒.

3) 目标时间频率(Frequency at target time), 在目标时间的频率为一标量或向量值. 缺省为 1Hz.

(5) 模块特点

1) 采样时间连续;

2) 参数有标量扩展;

3) 可向量化;

4) 没有过零区间.

7.1.3 Clock(时钟)

(1) 模块功能

显示并且提供仿真时间.

(2) 模块说明

Clock 模块在每一仿真步, 输出当时的仿真时间. 当该模块被打开时, 这一时间将显

示在窗口中. 在打开该模块的情况下运行仿真时会减慢仿真速度. Clock 模块对一些需要仿真时间的模块来说是很有用处的.

在离散系统中需要当前的时间时,使用 Digital Clock 模块.

(3) 模块数据类型

该模块输出为双精度类型实数值信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.3 所示.

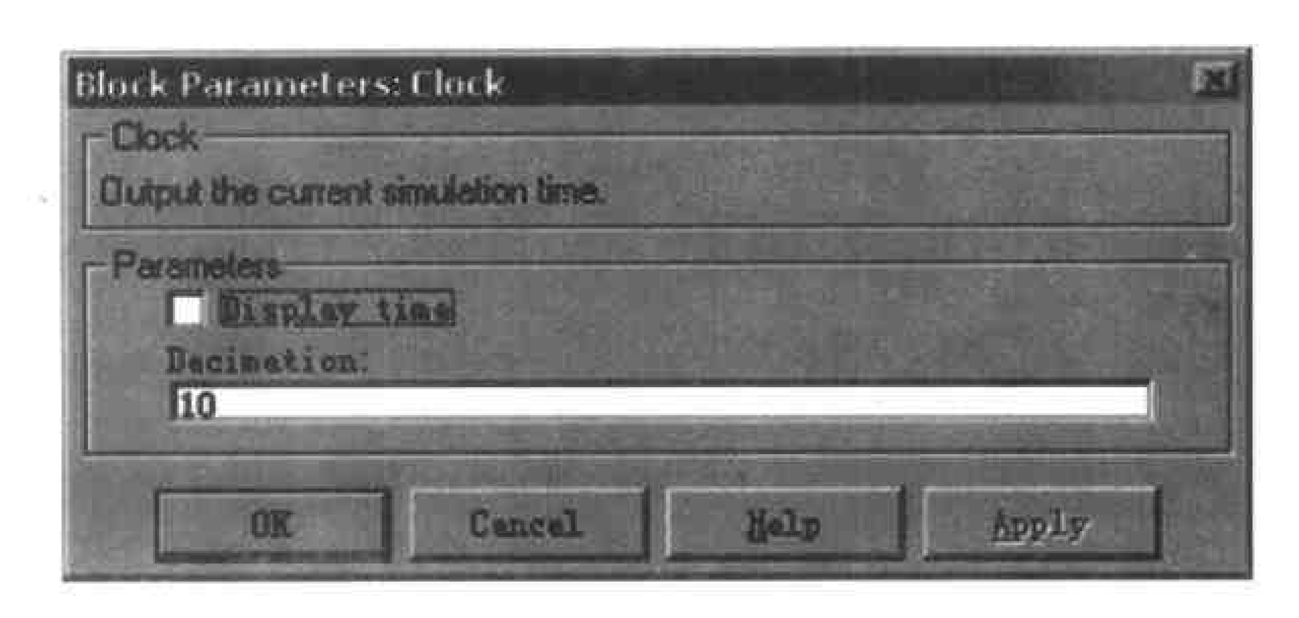


图 7.3 Clock 模块的参数对话框

1) 显示时间(Display time),使用该复选框,在模块图标旁显示当前仿真时间,并改变其图标的外观.

2) 抽取(Decimation),该参数值是时间更新获取的增量,可以是任意正整数,如取其值为 1000,则一个固定积分步为 1 毫秒,时钟将在 1 秒、2 秒等时刻更新.

(5) 模块特点

1) 采样时间连续;

2) 标量扩展不适用;

3) 不可向量化;

4) 没有过零区间.

7.1.4 Constant(常量)

(1) 模块功能

生成一个常量.

(2) 模块说明

Constant 模块生成一个与时间无关的指定的实数或复数值. 模块生成一个输出,它可以是标量也可以是向量,这取决于 Constant value 参数的长度. 模块的图标显示了指定的一个或多个常量值.

(3) 模块数据类型

该模块输出信号的数值类型与模块参数 Constant value 一样.

(4) 模块参数对话框

该模块的参数和对话框如图 7.4 所示.

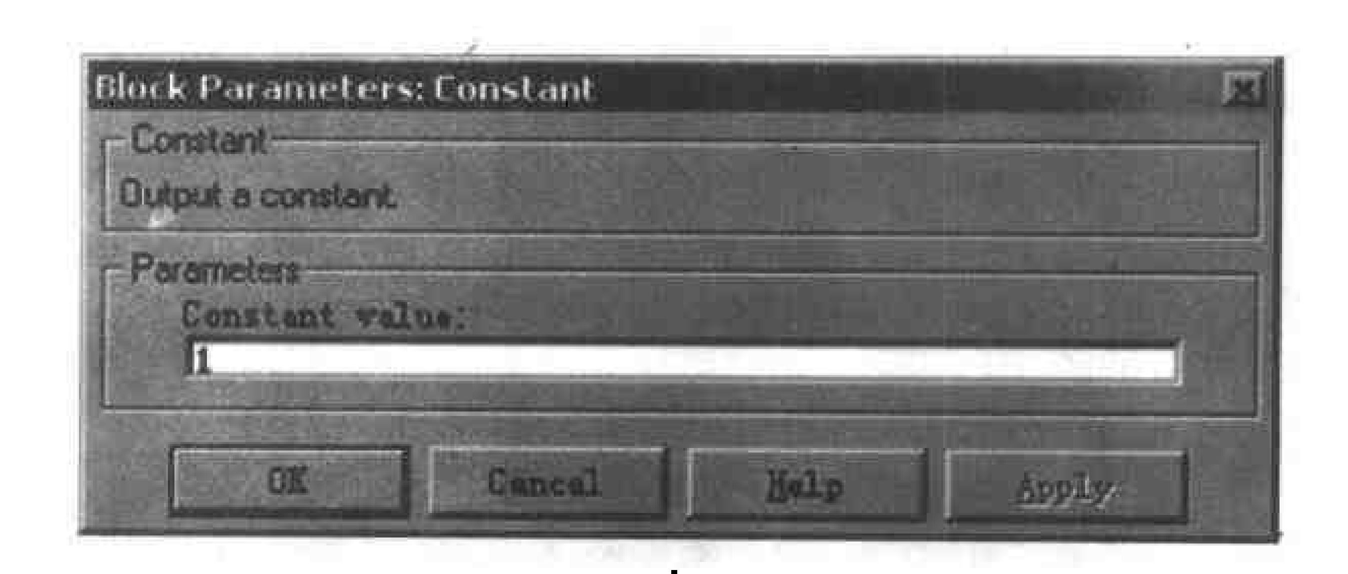


图 7.4 Constant 模块的参数对话框

常数值(Constant value)参数是该模块的输出,如果是一个向量,则输入为指定值的常数向量. 缺省值为 1.

(5) 模块特点

1) 采样时间为常数;

2) 标量扩展不适用;

3) 不可向量化;

4) 没有过零区间.

7.1.5 Digital Clock(数字时钟)

(1) 模块功能

以指定采样时间间隔输出仿真时间.

(2) 模块说明

Digital Clock 模块仅仅输出指定采样时间间隔的仿真时间. 在其它时间,输出保持为先前的值.

在离散系统中需要当前时间时,应使用该模块而不是 Clock 模块(输出连续时间).

(3) 模块数据类型

该模块输出为双精度类型实数值信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.5 所示.



图 7.5 Digital Clock 模块参数对话框

采样时间(Sample time)参数是指采样间隔. 缺省值为 1 秒.

(5) 模块特点

1) 采样时间离散;

2) 标量扩展不适用;

3) 不可向量化;

4) 没有过零区间.

7.1.6 Discrete Pulse Generator(离散脉冲生成器)

(1) 模块功能

生成间隔恒定的脉冲.

(2) 模块说明

Discrete Pulse Generator 模块以一定的时间间隔产生一系列的脉冲.

脉冲的宽度是脉冲为高电平时的采样周期的个数. 周期是脉冲为一个高电平和一个低电平时采样周期的个数. 相延迟是在脉冲开始前采样周期的数目. 相延迟可以是正数也可以是负数但不能大于周期. 采样时间必须大于 0.

在离散和混合系统中可以使用 Discrete Pulse Generator 模块,而如果要产生一个连续信号,应使用 Pulse Generator 模块.

(3) 模块数据类型

该模块接受和输出双精度类型实数值信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.6 所示.

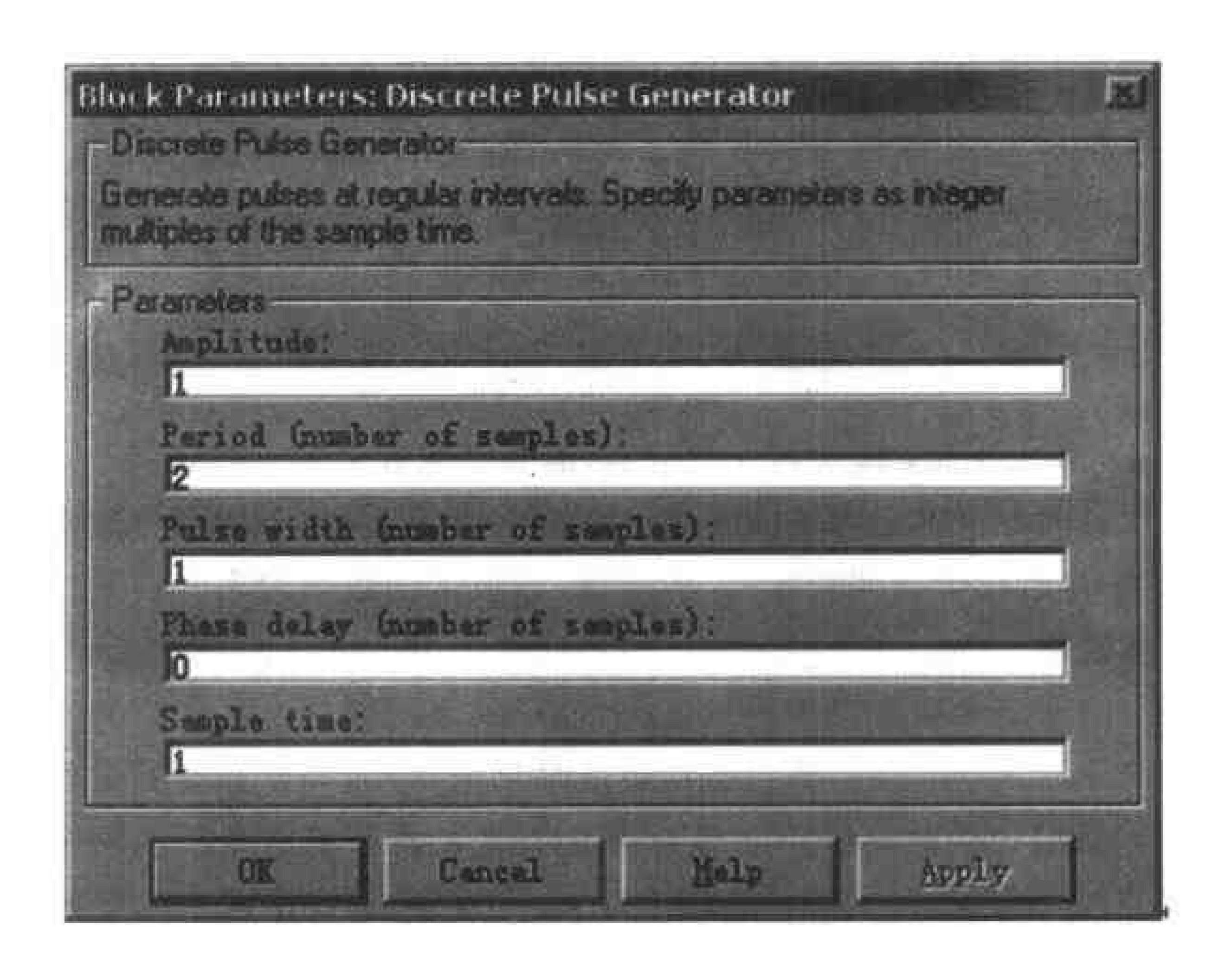


图 7.6 Discrete Pulse Generator 模块参数对话框

1）幅度（Amplitude），指发生脉冲的幅度. 其缺省值为 1.

2）周期（Period），以采样数为脉冲周期. 缺省值为 2.

3）脉冲宽度（Pulse width），脉冲为高电平的采样周期数. 缺省值为 1.

4）相延迟（Phase delay），每个脉冲发生前的延迟采样周期数. 缺省值为 0.

5）采样时间（Sample time），即采样周期. 缺省值为 1.

（5）模块特点

1）采样时间离散；

2）参数有标量扩展；

3）可向量化；

4）没有过零区间.

7.1.7 From Workspace（从工作空间读取数据）

（1）模块功能

从在工作空间中读取数据.

（2）模块说明

From Workspace 模块从 MATLAB 工作空间中读取数据. 模块的 Data 参数指定工作空间中的数据，由包含信号值和时间步表的矩阵或结构的 MATLAB 计算表达式来指定. 其矩阵或结构的格式与工作空间中输入的数据是一致的. 模块的图标显示了 Data 参数的表达式.

如果输入表没有指定输入数据值的次数，每个值假定在 t=(n-1) * st 时发生，其中：n 为第 n 个输入值，st 为模块的采样时间.

矩阵必须包含有两列以上的数据. 第一列应是单调递增的时间点. 另外的列应是与各行时间点对应的数据点. 矩阵应该有如式(7.2)的形式

$$\begin{bmatrix} t_1 & u1_1 & \cdots & un_1 \\ t_2 & u1_2 & \cdots & un_2 \\ \vdots & \vdots & \ddots & \vdots \\ t_{final} & u1_{final} & \cdots & un_{final} \end{bmatrix} \tag{7.2}$$

要读取由 To Workspace 模块写入的数据，需要在矩阵中加入时间.

该模块使用时间数据确定它的输出，但是它并不输出时间. 这就意味着对于一个包含 n 列的矩阵，模块输出一个长度为 n-1 的向量，它包含的数据为矩阵中某一行的除第一列外的其它所有数据.

如果在某一时间点上需要输出一个数，而该时间点的值是在工作空间的两个时间值之间，在选择 Interpolate data 时，该值将经过线性插值得到；如果需要的时间小于第一个时间值或者大于最后一个时间值，Simulink 使用开头的两个或者最后的两个点进行外推以得到所需要的时间值. Interpolate data 和 Hold final data value 两个参数的对输出的影响如表 7.2 所示，表中 ti 表示初始（第一个）时间，tf 表示结束（最后一个）时间.

表 7.2 模块输出与两个参数选项的关系

Interpolate data 选项	Hold final data value 选项	$t_i<t<t_f$ 时的模块输出	$t>t_f$ 时模块输出
On	Off	在数据值之间插值	从最后值外推
On	On	在数据值之间插值	最后值
Off	Off	最近的数据值	0
Off	On	最近的数据值	最后值

如果输入表中在同一时间点上包含有多个输入数据，输出是最后遇到的那个数据.

(3) 模块数据类型

该模块可以输出任何数据类型实数或复数值信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.7 所示.

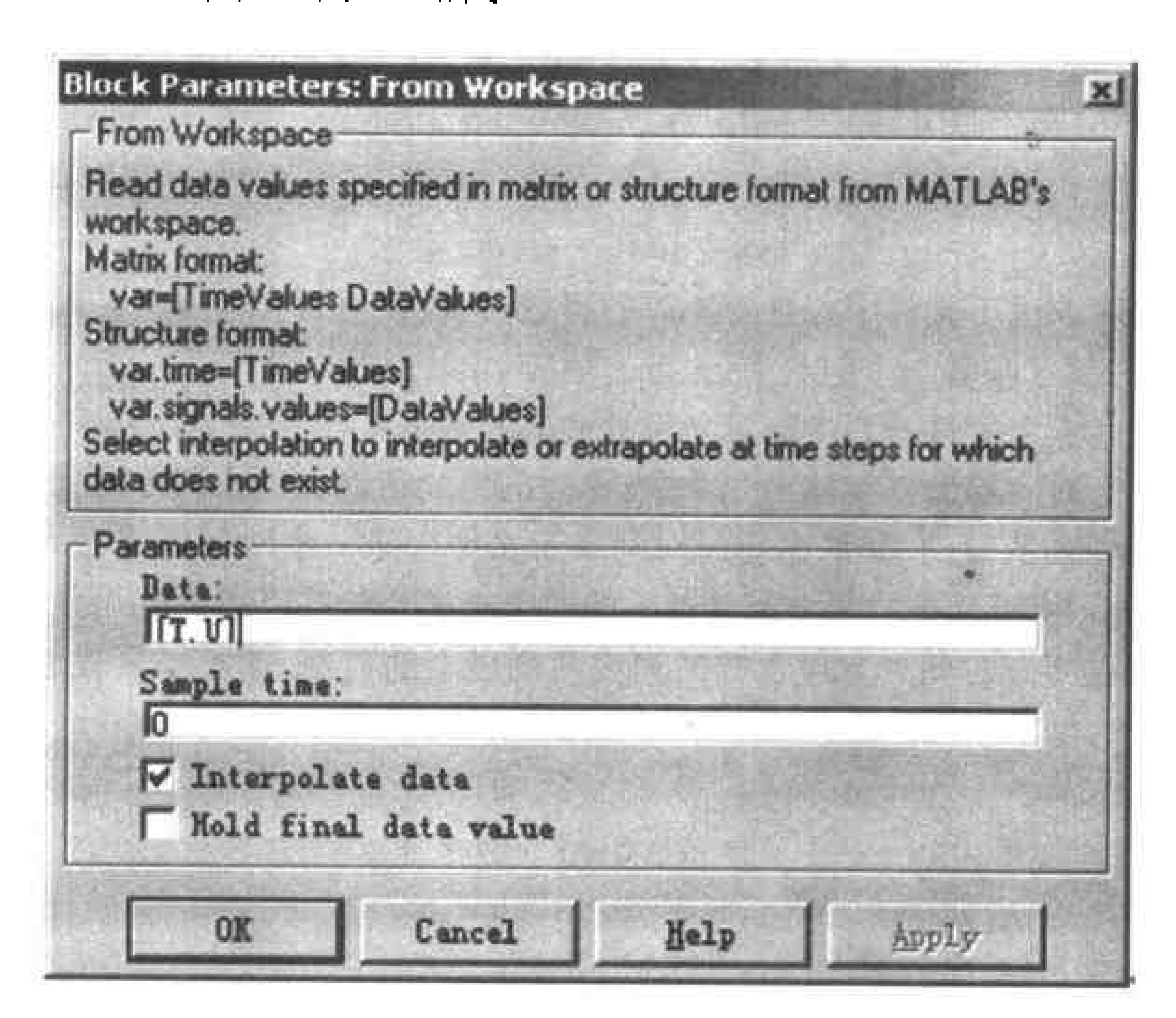


图 7.7 From Workspace 模块参数对话框

1) 数据(Data)是一个包含有仿真时间和相应信号值表，计算到矩阵或结构的表达式. 假设工作空间包含有时间 T 的列向量和相应信号值的矩阵 U，则缺省表达式为 [T,U]，这就产生了一个包含所需输入表的矩阵. 如果所需的信号-时间矩阵或结构已经存在于工作空间中，则只要在该域中输入矩阵或结构的名字即可.

2) 采样时间(Sample time)从工作空间读入数据的采样率.

3) 数据插值(Interpolate data)是使模块进行线性插值(或外推)的选项.

4) 保持最终数据值(Hold final data value)，该选项使模块输出保持最后可用的值.

(5) 模块特点

1) 采样时间从驱动模块继承；

2）标量扩展不适用；

3）可向量化；

4）没有过零区间.

7.1.8 From File（从文件读数据）

（1）模块功能

从文件读数据.

（2）模块说明

From File 模块从指定的文件读取数据作为其输出. 模块的图标中显示了提供数据的文件名.

文件必须包含有一个两行或两行以上的矩阵. 第一行应是单调递增的时间点. 其它行应是与各列时间点相对应的数据点. 矩阵应该具有如下的形式：

$$\begin{bmatrix} t_1 & t_2 & \cdots & t_{\text{final}} \\ ul_1 & ul_2 & \cdots & ul_{\text{final}} \\ \vdots & \vdots & \ddots & \vdots \\ un_1 & un_2 & \cdots & un_{\text{final}} \end{bmatrix} \tag{7.3}$$

输出的宽度取决于文件中包含的矩阵的行数. 模块使用时间数据确定其输出，但是不输出时间值. 这就意味着一个包含有 m 行的矩阵，模块输出一个长度为 m－1 的向量，它包含有对应时间点上的这一列中除第一行外的其它数据.

如果在某一时间点上需要输出一个数，该时间点的值在文件中两个时间点的值之间，该值将经过线性插值得到. 如果需要的时间小于文件中第一个时间值或者大于最后一个时间值，Simulink 使用开头的两个或者最后的两个点进行外推以得到所需要的时间值.

如果矩阵在同一时间点上包含有两列或两列以上的数据，输出是最先遇到的那一列上的数据. 例如，如果一个矩阵包含如下的数据：

$$\begin{bmatrix} 0 & 1 & 2 & 2 \\ 2 & 3 & 1 & 2 \\ 3 & 0 & 4 & 5 \end{bmatrix}$$

在时间点 2 上，输出数据为[1;4].

可以使用 To File 或者 To Workspace 模块保存数据. 由 To File 模块写入的数据，From File 模块能够不加修改地读取. 而由 To Workspace 模块写入并保存在文件上的数据，要读取它就必须有以下要求：

1）数据必须包含有仿真时间. 在仿真的输出中，将时间数据包含进去的最简单的方法，是在 Simulation Parameters 对话框中的 Workspace I/O 页中，为时间指定一个变量.

2）写入工作空间的数据的形式与 From File 模块读取的数据形式不一样. 在保存数据到文件之前，必须先转换数据的位置. 这时数据由 From File 模块读取的时候，它才会以正确的形式给出.

（3）模块数据类型

该模块输出为双精度类型实数值信号.

（4）模块参数对话框

该模块的参数对话框如图 7.8 所示.

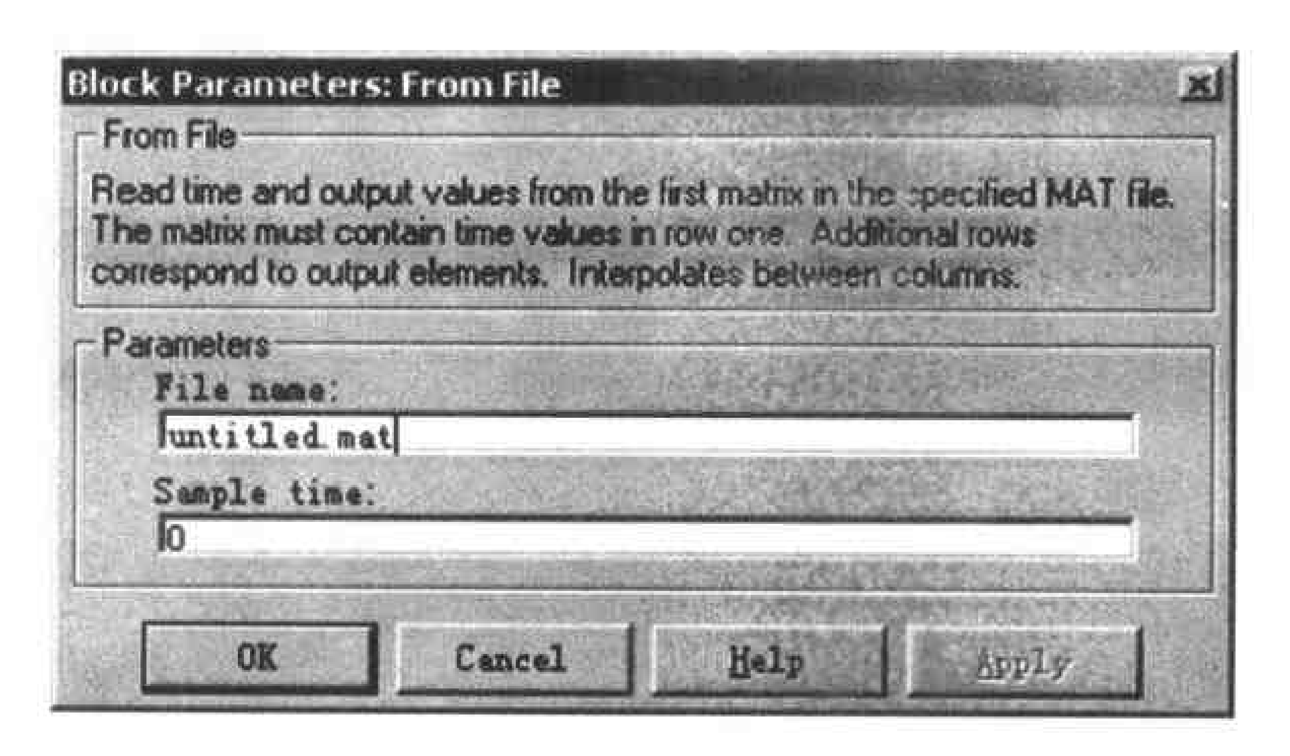


图 7.8 From File 模块参数对话框

1）来自文件名(From name)，包含用于输入数据的文件名. 缺省值为：untitled. mat.

2）采样时间(Sample time)，从文件中读取数据的采样率.

(5) 模块特点

1）采样时间从驱动模块继承；

2）标量扩展不适用；

3）可向量化；

4）没有过零区间.

7.1.9 Pulse Generator(脉冲生成器)

(1) 模块功能

以一定的间隔生成脉冲.

(2) 模块说明

Pulse Generator 模块以一定的间隔产生一系列的脉冲. 在连续系统中使用 Pulse Generator 模块. 在离散系统中使用 Discrete Pulse Generator 模块.

(3) 模块数据类型

该模块输出为双精度类型实数值信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.9 所示.

1）周期(Period)，以秒计的脉冲周期. 缺省值为 1 秒.

2）任务周期(Duty cycle)，信号为开占脉冲周期的百分数. 缺省值为百分之五十.

3）幅度(Amplitude)，脉冲的幅度. 缺省值为 1.

4）开始时间(Start time)，脉冲开始产生之前的延迟，以秒为单位. 缺省值为 0 秒.

(5) 模块特点

1）采样时间是继承的；

2）参数有标量扩展；

3）可向量化；

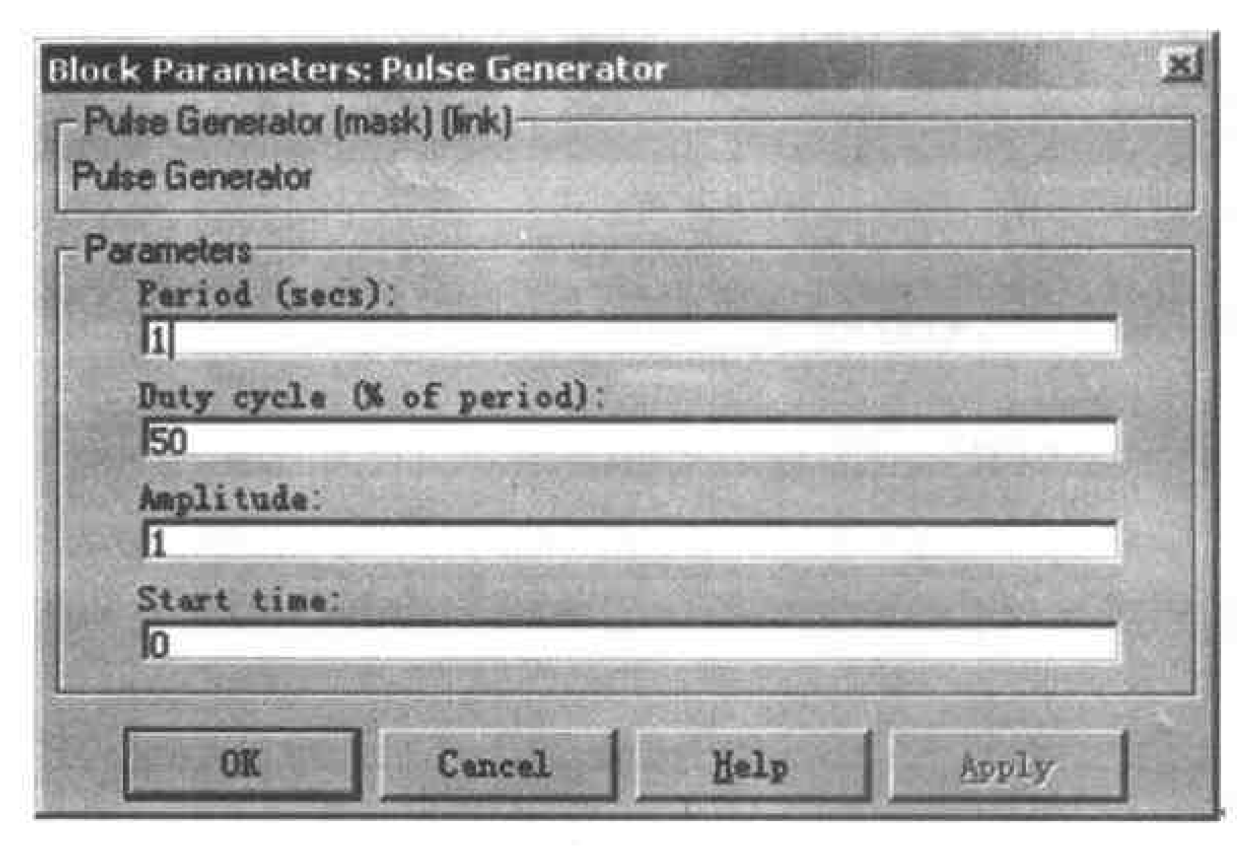


图 7.9 Pulse Generator 模块参数对话框

4）没有过零区间.

7.1.10 Ramp（倾斜）

（1）模块功能

产生连续增大或者减小的信号.

（2）模块说明

Ramp 模块生成一从指定的时间和大小开始，以一定的速率增大或减小的信号.

（3）模块数据类型

该模块输出双精度类型信号.

（4）模块参数对话框

该模块的参数对话框如图 7.10 所示.

1）斜率(Slope)，指产生信号的变化率. 缺省值为 1.

2）开始时间(Start time)，指信号开始产生的时间. 缺省值为 0.

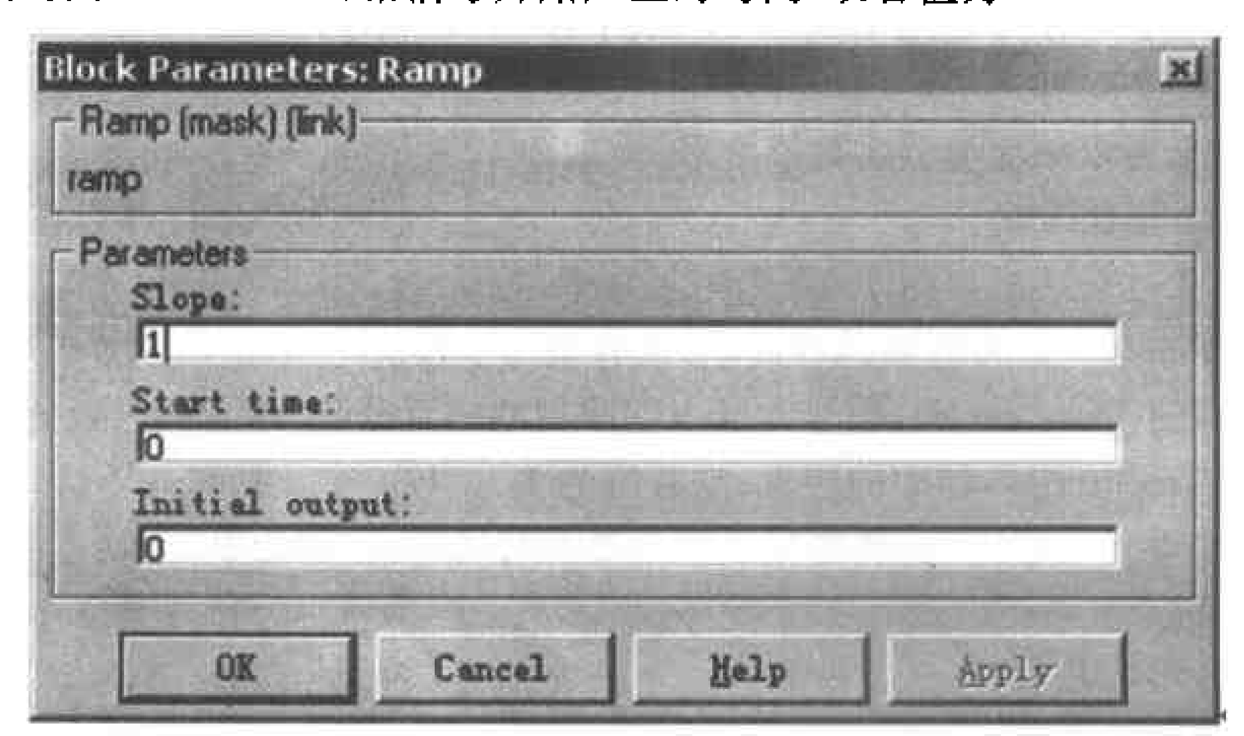


图 7.10 Ramp 模块参数对话框

3）初始输出（Initial output），指信号的初值. 缺省值为 0.

（5）模块特点

1）采样时间从驱动模块继承；

2）有标量扩展；

3）可向量化；

4）有过零区间.

7.1.11 Random Number（随机数产生器）

（1）模块功能

产生正态分布的随机数.

（2）模块说明

Random Number 模块生成正态分布的随机数. 每次仿真开始种子被置指定值.

缺省时，产生的随机数序列的均值为 0，方差为 1，可以改变这些参数. 产生的随机数序列是可重复的并且能够用任何 Random Number 模块以相同的参数产生. 要生成相同均值和方差的随机数向量，指定 Initial seed 参数为向量即可.

要生成均匀分布的随机数，使用 Uniform Random Number 模块.

应当避免对一个随机信号进行积分，因为求解器对光滑信号进行积分才有意义. 此时，可用 Band-Limited White Noise 模块代替.

（3）模块数据类型

该模块接受和输出双精度类型信号.

（4）模块参数对话框

该模块的参数对话框如图 7.11 所示.

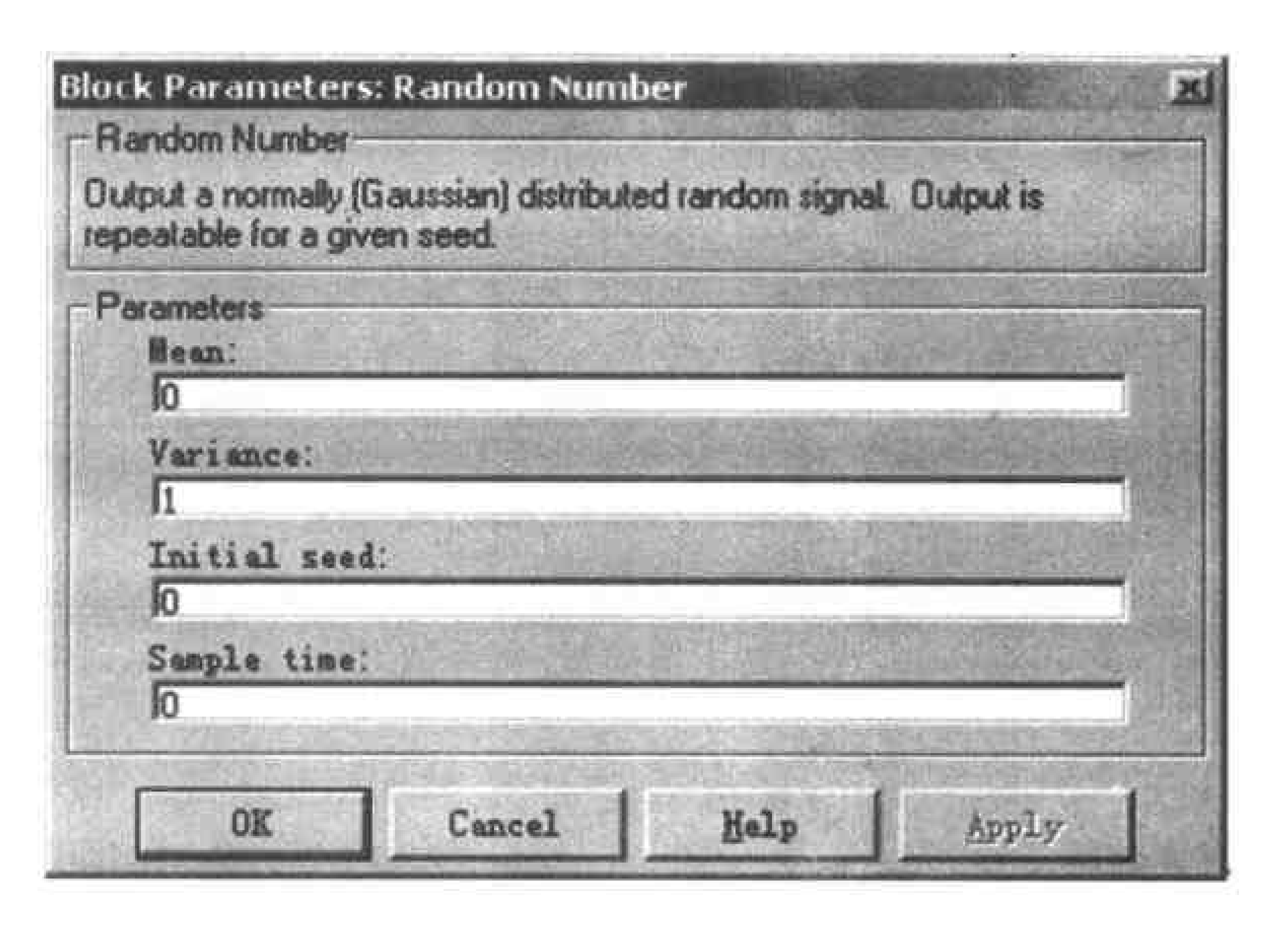


图 7.11 Random Number 模块参数对话框

1）均值（Mean），指产生随机数的均值. 缺省值为 0.

2）方差（Variance），指产生随机数的方差. 缺省值为 1.

3）初始种子(Initial seed)，指随机数发生器开始的种子.缺省值为 0.

4）采样时间(Sample time)，指采样时间间隔.缺省值为 0，使模块具有连续采样时间.

（5）模块特点

1）采样时间连续或离散；

2）参数有标量扩展；

3）可向量化；

4）没有过零区间.

7.1.12 Repeating Sequence(重复序列)

（1）模块功能

产生重复的任意信号.

（2）模块说明

通过 Repeating Sequence 模块可以生成随着时间的推移在波形上重复的信号，波形可任意指定.当仿真达到 Time values 向量中的最大时间值时信号开始重复.

该模块是使用一维 Look-Up Table 模块实现的，在各个点之间进行了线性插值.

（3）模块数据类型

该模块输出双精度类型实数值信号.

（4）模块参数对话框

该模块的参数对话框如图 7.12 所示.

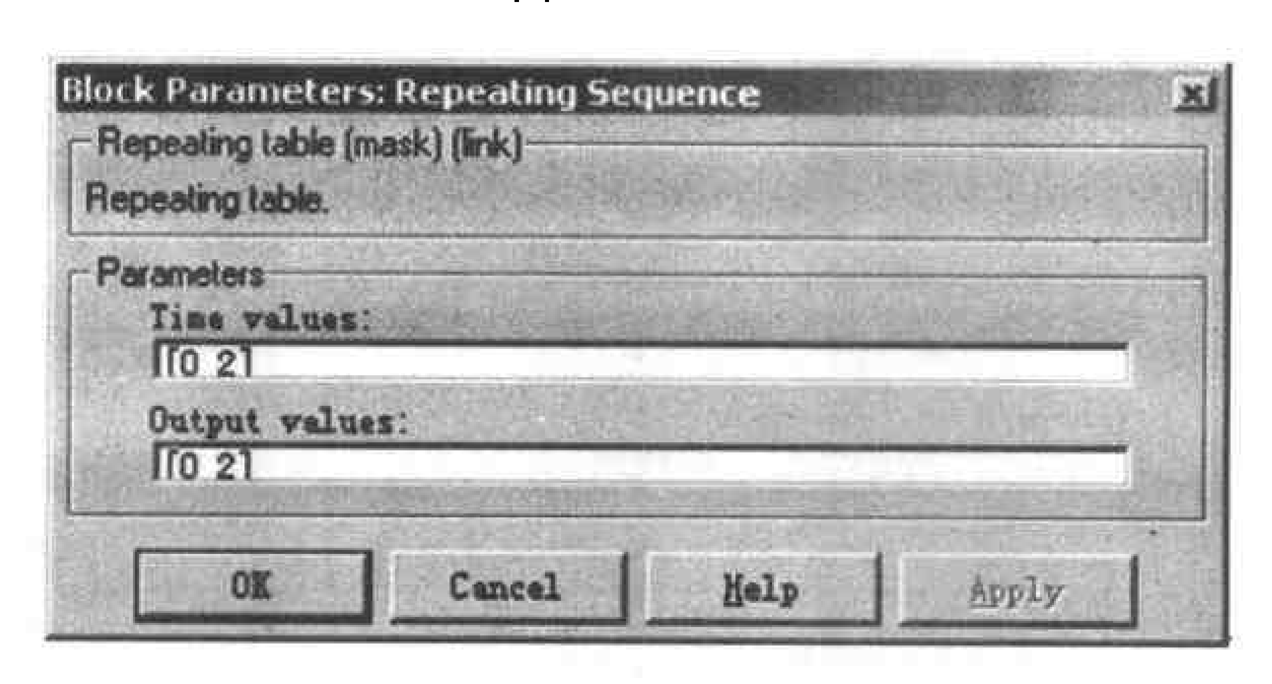


图 7.12 Repeating Sequence 模块参数对话框

1）时间值(Time values)，是一单调增加的时间值向量.缺省为：[0 2].

2）输出值(Output values)，为一输出值向量.每个值对应同一时间列中的时间值.缺省为[0 2].

（5）模块特点

1）采样时间连续；

2）标量扩展不适用；

3）不可向量化；

4）没有过零区间.

7.1.13 Signal Generator(信号发生器)

(1) 模块功能

生成不同的波形.

(2) 模块说明

Signal Generator 模块能够产生三种不同的波形:正弦波、方波和锯齿波.信号的参数可以用 Hz 或者弧度每秒为单位来描述.

负的 Amplitude 参数值会产生一个 180°的相移,可以通过多种方法产生一个不是 180°相移的波形,包括输入 Clock 模块的信号给 MATLAB Fcn 模块,并且给特定的波形编写方程式.

当仿真正在进行时,可以改变 Signal Generator 模块的输出设置.这对于快速确定系统对不同类型输入的响应是非常有用的.

(3) 模块数据类型

该模块输出双精度类型实数值信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.13 所示.

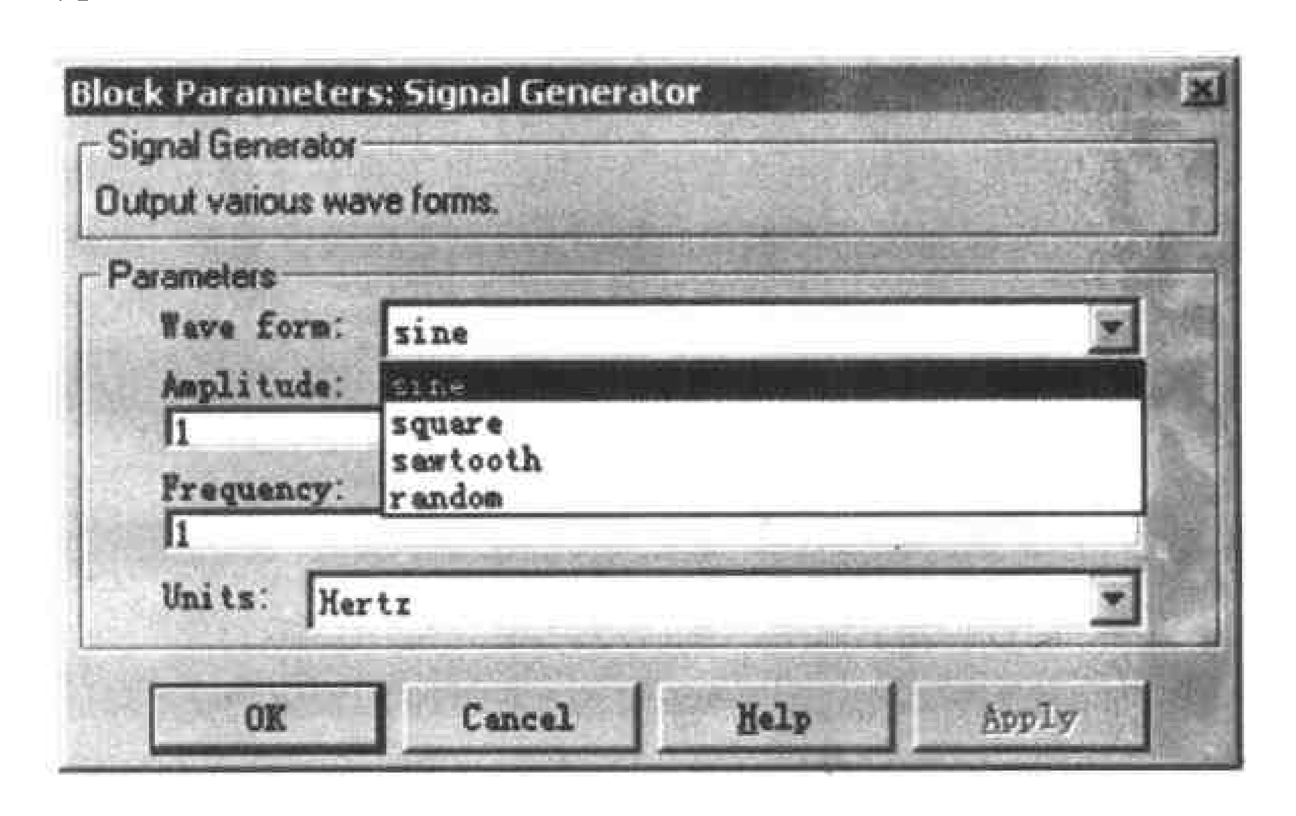


图 7.13 Signal Generator 模块参数对话框

1) 波形(Wave form),包括:正弦波(sine wave),方波(square wave)和锯齿波(sawtooth wave).缺省为:正弦波.

2) 幅度(Amplitude),指产生信号的幅度.缺省值为 1.

3) 频率(Frequency),指产生信号的频率.缺省为 1.

4) 单位(Units),指信号频率单位,有 Hertz 或 rad/sec 两种.缺省值为 Hertz.

(5) 模块特点

1) 继承采样时间;

2) 参数有标量扩展;

3) 可向量化;

4）没有过零区间.

7.1.14　Sine Wave（正弦波）

（1）模块功能

产生一个正弦波.

（2）模块说明

Sine Wave 模块提供正弦曲线. 该模块既能够提供连续形式的正弦波也能够提供离散形式的正弦波.

Sine Wave 模块的输出由下式确定：

输出＝幅度×sin（频率×时间＋相位）

Sample time 参数的值确定模块是工作于连续模式还是离散模式：

0　　缺省值，使模块工作于连续模式；

＞0　　使模块工作于离散模式；

－1　　使模块的工作模式与接收信号的模块相同.

1）使用离散模式下的 Sine Wave 模块. 大于 0 的 Sample time 参数值使得该模块就像是由采样时间被设为该值的 Zero-Order Hold 模块驱动的一样.

使用这种方式的 Sine Wave 模块，允许创建纯离散的正弦波信号源的模型，而不是由连续和离散组成的混合模型. 混合系统更复杂，因而仿真需要的时间更长.

离散形式的 Sine Wave 模块使用的是增量算法，而不是基于绝对时间的算法. 因此，该模块就能够用于试图长时间运行的模型，诸如振动或者疲劳试验.

增量算法基于前一采样时间的计算结果计算正弦波. 该方法的计算公式如式(7.4)所示.

$$\begin{aligned}\sin(t+\Delta t) &= \sin(t)\cos(\Delta t)+\sin(\Delta t)\cos(t)\\ \cos(t+\Delta t) &= \cos(t)\cos(\Delta t)-\sin(t)\sin(\Delta t)\end{aligned} \tag{7.4}$$

式(7.4)可以写成式(7.5)的矩阵形式：

$$\begin{bmatrix}\sin(t+\Delta t)\\ \cos(t+\Delta t)\end{bmatrix}=\begin{bmatrix}\cos(\Delta t) & \sin(\Delta t)\\ -\sin(\Delta t) & \cos(\Delta t)\end{bmatrix}\begin{bmatrix}\sin(t)\\ \cos(t)\end{bmatrix} \tag{7.5}$$

因为 Δt 是常数，因此式(7.6)是常量.

$$\begin{bmatrix}\cos(\Delta t) & \sin(\Delta t)\\ -\sin(\Delta t) & \cos(\Delta t)\end{bmatrix} \tag{7.6}$$

因此问题就变成，sin(t)乘以常量矩阵以得到 sin($t+\Delta t$)的矩阵乘法. 这一算法对于没有支持三角函数的浮点运算硬件的计算机来说，也是比较快的.

2）使用连续形式下的 Sine Wave 模块. 值为 0 的 Sample time 参数使得该模块运行在连续模式下. 当运行于连续模式时，Sine Wave 模块可能变得不精确，时间变得长时会损失精度.

（3）模块数据类型

该模块接受和输出双精度类型实数值信号.

（4）模块参数对话框

该模块的参数对话框如图 7.14 所示.

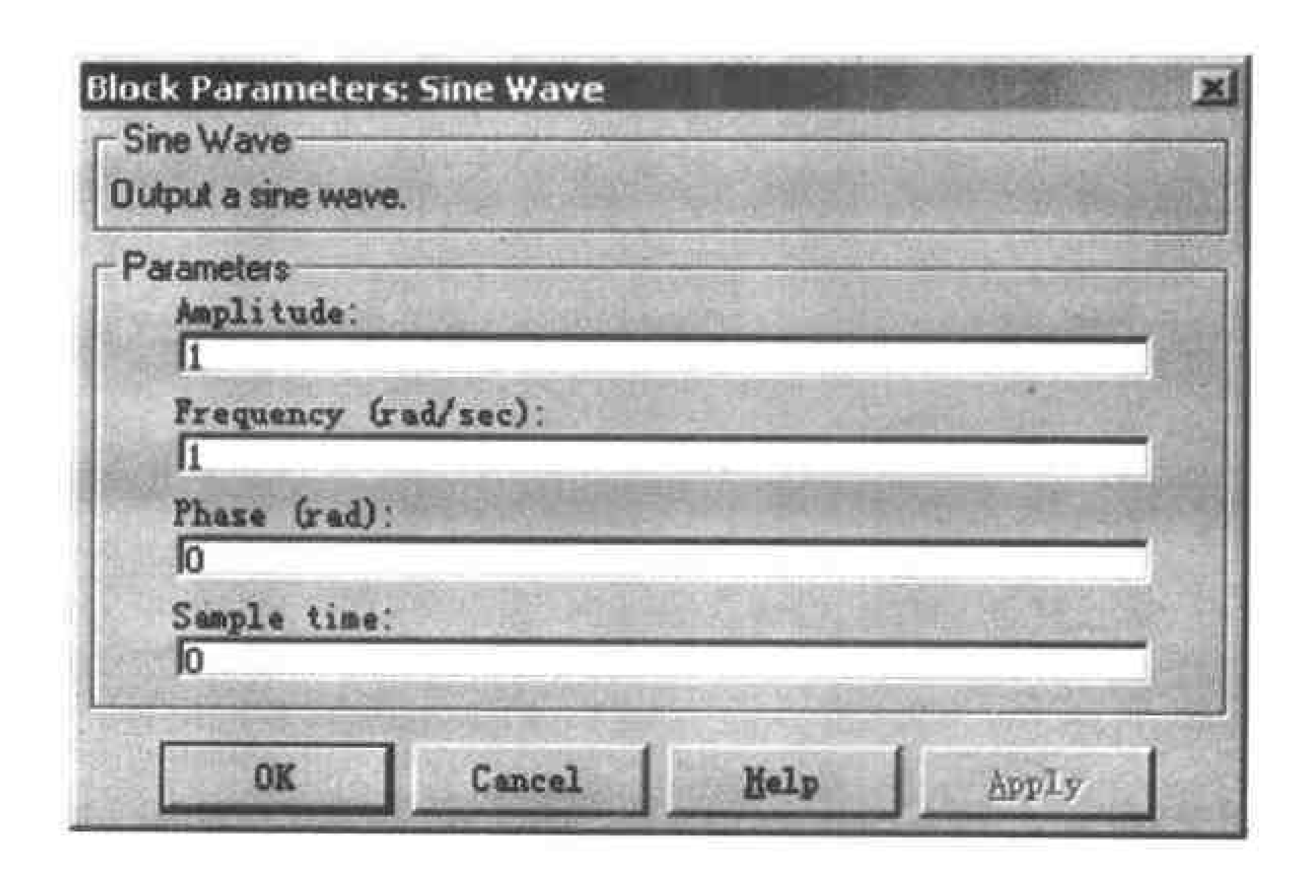


图 7.14　Sine Wave 模块参数对话框

1) 幅度(Amplitude),指信号的幅度.缺省值为 1.

2) 频率(Frequency),指信号的频率,以弧度/秒(rad/sec)为单位.

3) 相移(Phase),产生信号的相移,以弧度为单位.缺省值为 0.

4) 采样时间(Sample time),指采样周期.缺省值为 0.

(5) 模块特点

1) 采样时间连续、离散或继承;

2) 参数有标量扩展;

3) 可向量化;

4) 没有过零区间.

7.1.15　Step(阶跃)

(1) 模块功能

产生阶跃函数.

(2) 模块说明

Step 模块提供在指定的时间处在两个可定义的水平间的阶跃.如果仿真时间小于 Step time 参数值,模块的输出是 Initial value 参数的值,当仿真时间大于或者等于 Step time 时,输出是 Final value 参数的值.

Step 模块产生标量或者向量的输出,这取决于参数的长度.

(3) 模块数据类型

该模块输出双精度类型实数值信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.15 所示.

1) 阶跃时间(Step time),以秒为单位,指输出从初始值(Initial value)跳变至最终值(Final value)的时间.缺省值为 1 秒.

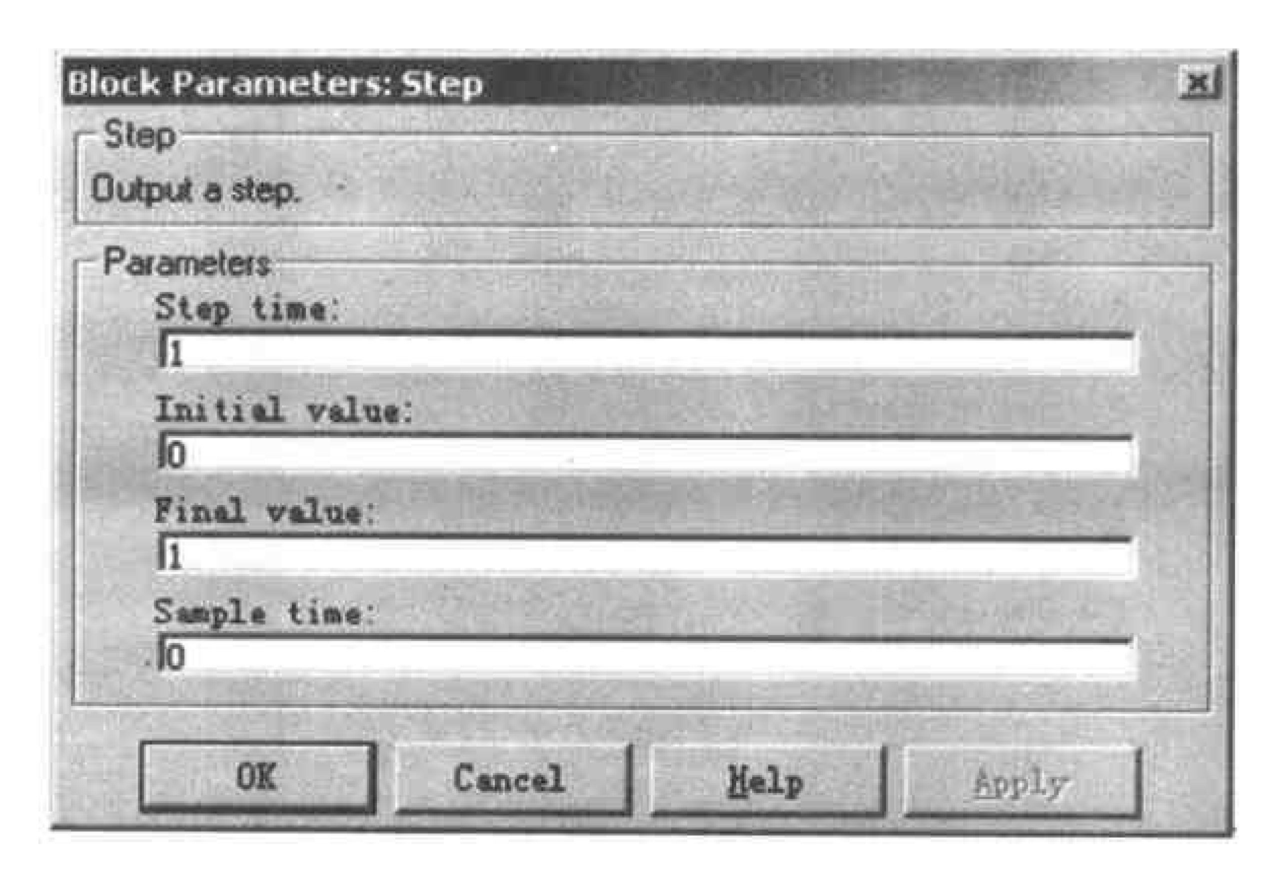


图 7.15　Step 模块参数对话框

2）初始值(Initial value)，指直到仿真时间到达阶跃时间(Step time)参数时的模块输出.缺省值为 0.

3）最终值(Final value)，指仿真时间到达并超过阶跃时间(Step time)参数时的模块输出.缺省值为 1.

4）采样时间(Sample time)，指阶跃采样率.

(5) 模块特点

1）采样时间由驱动模块继承；

2）参数有标量扩展；

3）可向量化；

4）有过零区间，检测阶跃时间.

7.1.16　Uniform Random Number（均匀分布随机数）

(1) 模块功能

生成均匀分布的随机数.

(2) 模块说明

Uniform Random Number 模块在指定的区间内，以指定的起始种子，生成均匀分布的随机数.在每次仿真开始时重新设置种子.生成的随机序列是可重复的并且能够由 Uniform Random Number 模块以相同的种子和参数生成.要生成随机数向量，指定 Initial seed 参数为向量即可.

要生成正态分布的随机数，使用 Random Number 模块.

避免对随机信号进行积分，因为求解器积分相对光滑的信号才有意义.此时采用 Band-Limited White Noise 模块.

(3) 模块数据类型

该模块输出双精度类型实数值信号.

（4） 模块参数对话框

该模块的参数对话框如图 7.16 所示.

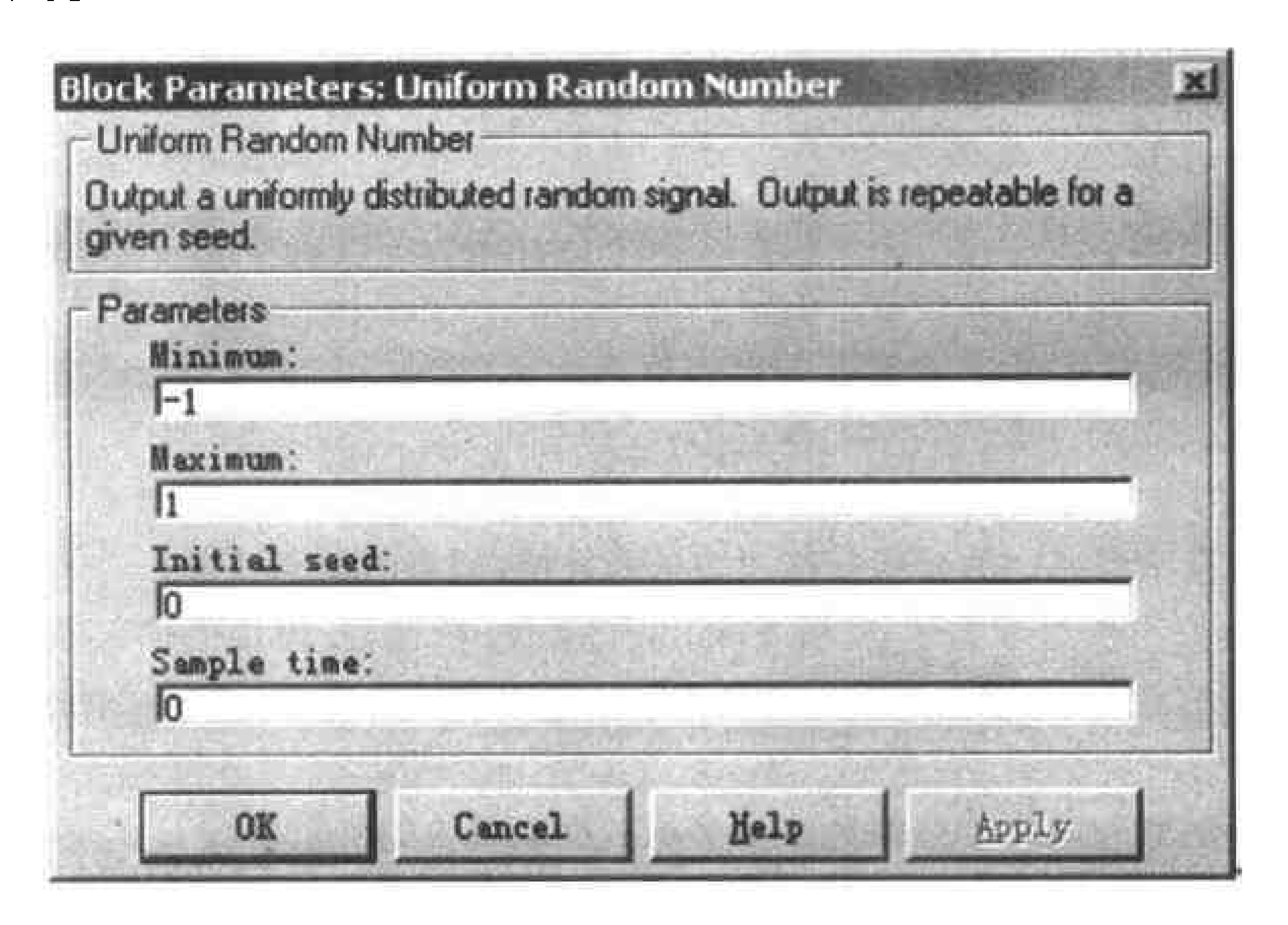


图 7.16　Uniform Random Number 模块参数对话框

1） 最小值(Minimum)，指定区间的最小值. 缺省值为－1.

2） 最大值(Maximum)，指定区间的最大值. 缺省值为 1.

3） 初始种子(Initial seed)，指随机数产生的开始种子. 缺省值为 0.

4） 采样时间(Sample time)，指采样周期. 缺省值为 0.

（5） 模块特点

1） 采样时间连续、离散或继承；

2） 标量扩展不适用；

3） 可向量化；

4） 没有过零区间.

7.2　Sinks 库中的模块

Sinks 库中包含的模块如表 7.3 所列，各个模块的图标如图 6.8 所示.

表 7.3　Sinks 库中的模块

模块名	功　能
Display	显示输入的值
Scope	显示仿真期间产生的信号
Stop Simulation	当输入为非零时停止仿真
To File	向文件中写数据
To Workspace	向工作空间中的矩阵写入数据
XY Graph	使用 MATLAB 的图形窗口显示信号的 X-Y 图

7.2.1 Display(显示)

(1) 模块功能

显示输入的值.

(2) 模块说明

Display 模块显示输入的值.可以通过选择 Format 选项来控制显示的格式:

1) short,显示缩放的 5 位数字的定点值;

2) long,显示缩放的 15 位数字的定点值;

3) short_e,显示有 5 位数字的浮点值;

4) long_e,显示有 16 位数字的浮点值;

5) bank,显示用固定的元和分格式表示的数值(但是没有 $ 符号).

要将该模块用作浮动显示,选择 Floating display 核选框.模块的输入端口会消失,模块显示被选中的连线上的信号的值.选择 Floating display 核选框,必须关闭 Simulink 缓存再利用特性.

数据显示的数量和在哪些时间步上显示数据由模块的参数确定:

1)通过设定 Decimation 参数为 n,可以每 n 个采样显示一个数据,n 是降采样因子.缺省的降采样因子是 1,这时在每一时间步都显示数据.

2)通过 Sample time 参数可以指定显示数据的时间间隔.这一参数对于使用变步长的求解器非常有用,因为它们的时间步之间的时间间隔可能不相等.缺省值 -1 使得模块在确定哪些点要显示时不考虑采样间隔.

如果模块的输入是向量,可以改变模块图标的大小以使其显示的不仅仅是第一个元素.可以在垂直方向和水平方向上改变模块图标的大小,模块将在适当的方向增加显示区域.一个黑色的三角形表明模块没有显示出所有的向量元素.例如,图 7.17 所示的模型将一个向量传给了一个 Display 模块.图 7.17(a)的模型是 Display 模块的大小改变之前的样子,注意它有一个黑色的三角形,图 7.17(b)的模型显示了 Display 模块的大小被改变之后的样子,它将三个输入元素都显示出来了.

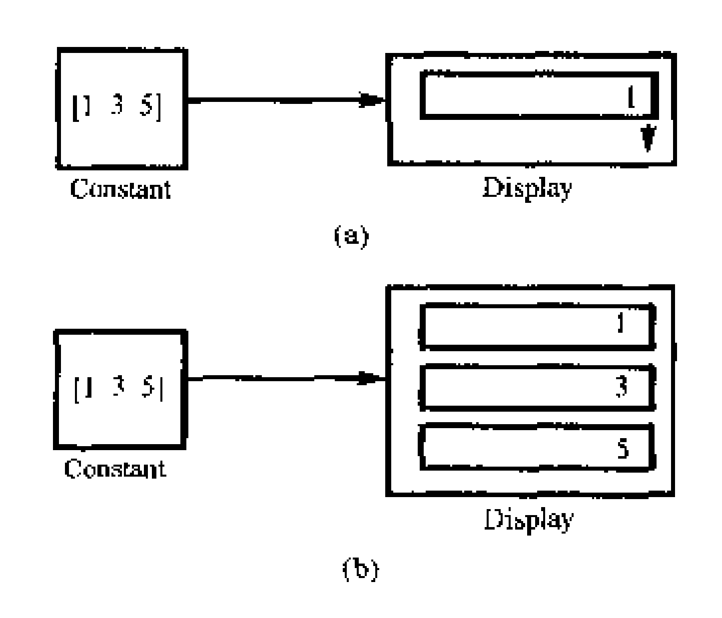


图 7.17 Display 模块不同大小的不同显示效果

(3) 模块数据类型.该模块接受和输出任意数据类型的实数或复数值信号.

(4) 模块参数对话框. 该模块的参数对话框如图 7.18 所示.

1) 格式(Format),指显示数据的格式. 缺省值为 short.

2) 抽取(Decimation),指显示数据的频度. 缺省值为 1,显示每个输入点.

3) 浮动显示(Floating display),如果选中此核选框,模块的输出端口消失,可以将此模块作为浮动显示模块.

4) 采样时间(Sample time),显示点采样时间.

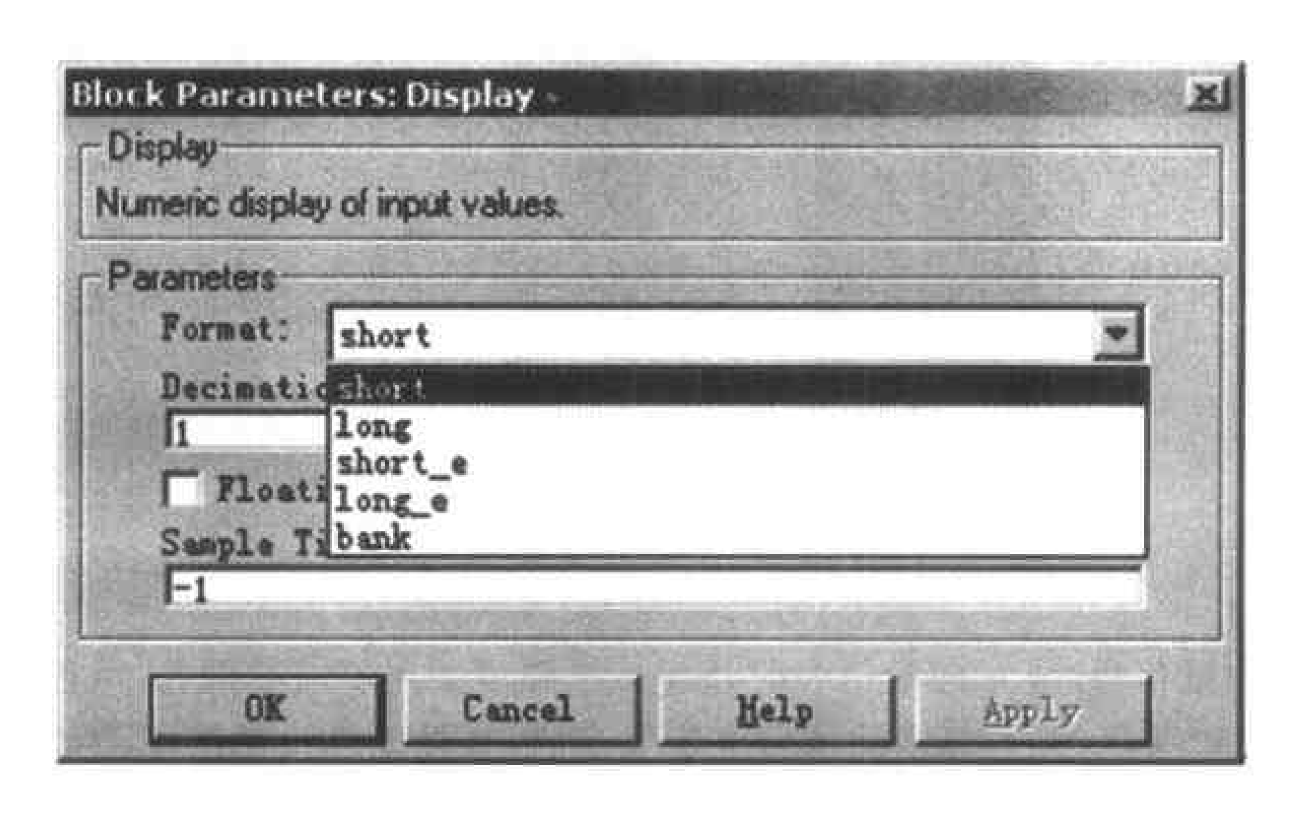


图 7.18 Display 模块参数对话框

(5) 模块特点

1) 采样时间由驱动模块继承;

2) 可向量化.

7.2.2 Scope(显示器)

(1) 模块功能

显示仿真时产生的信号.

(2) 模块说明

Scope 模块显示输入关于仿真时间的图形. 模块接收一个输入并且能够显示多个信号的图形. Scope 模块允许调整时间的大小和显示输入值的范围. 可以移动 Scope 窗口,也可以改变它的大小,还可以在仿真期间改变 Scope 的参数值.

在开始仿真时,Simulink 并不打开 Scope 窗口,尽管它传送数据给有关的 Scope. 因此,在仿真后,如果打开 Scope,Scope 的输入信号则会显示出来.

如果信号是连续的,Scope 生成由点连成的图形. 如果信号是离散的,Scope 生成阶梯图.

Scope 提供工具条按钮,可以缩放显示的数据;可以在 Scope 中显示所有的数据;可以将一个仿真中的坐标轴的设置保存给下一个仿真;可以限制显示的数据;可以保存数据到工作空间.

图 7.19 中标出了工具条按钮,它是打开 Scope 模块时 Scope 窗口所显示的样子.

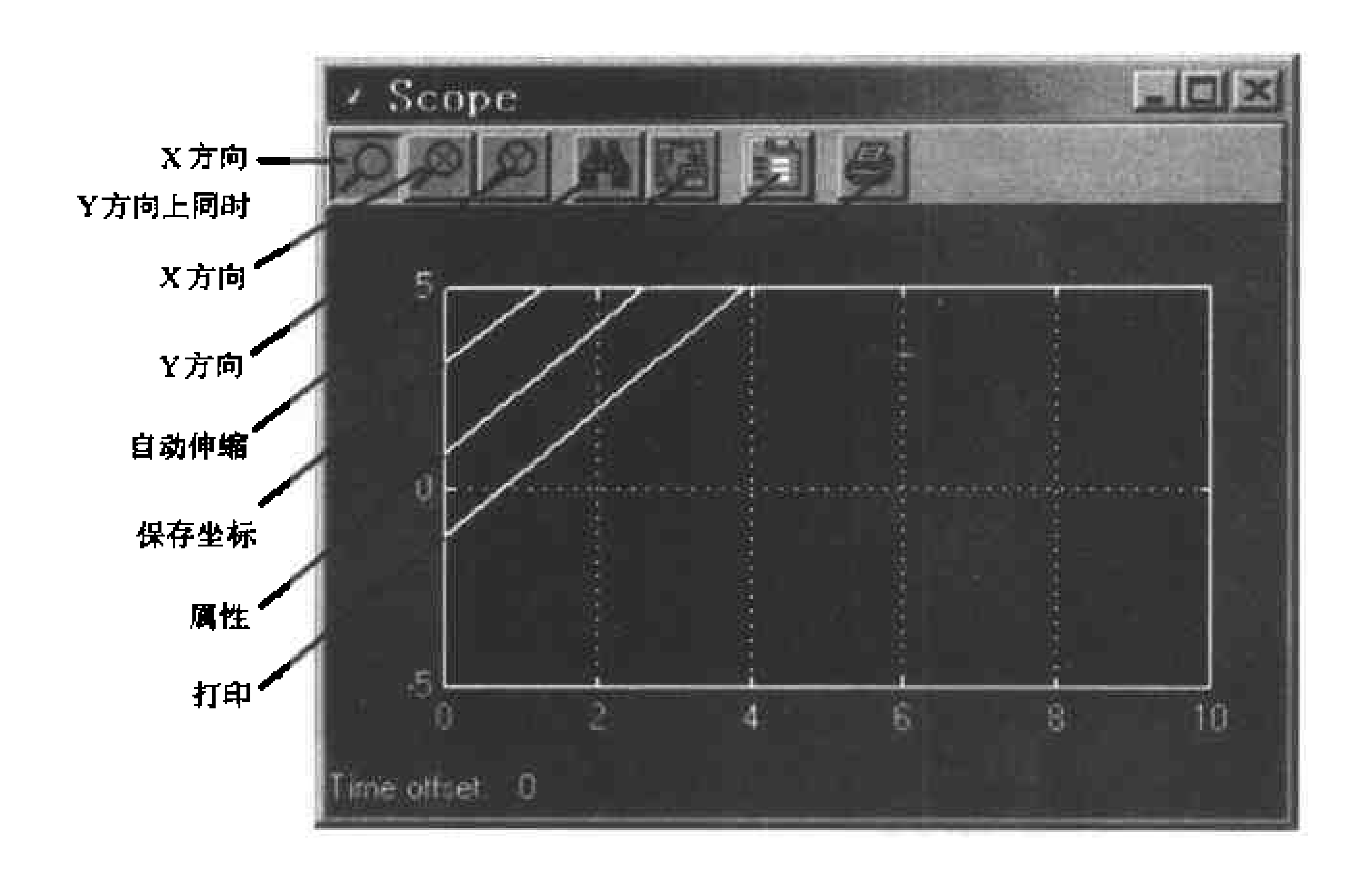


图 7.19　Scope 窗口及工具条按钮

1) 显示向量信号. 显示向量信号时,Scope 按顺序使用不同的颜色:黄色、洋红色、青色、红色、绿色和深蓝色. 当显示的信号多于 6 个时,按前面排出的顺序循环使用这些颜色.

2) Y 轴极限. 右击坐标轴并选择 Axes Properties... 选项,则显示图 7.20 所示的对话框,来设置 Y 轴极限. 其中,参数 Y-min,用来输入 Y 轴的最小值;Y-max,用来输入 Y 轴的最大值;Title,用来输入图的标题,是一个字符串.

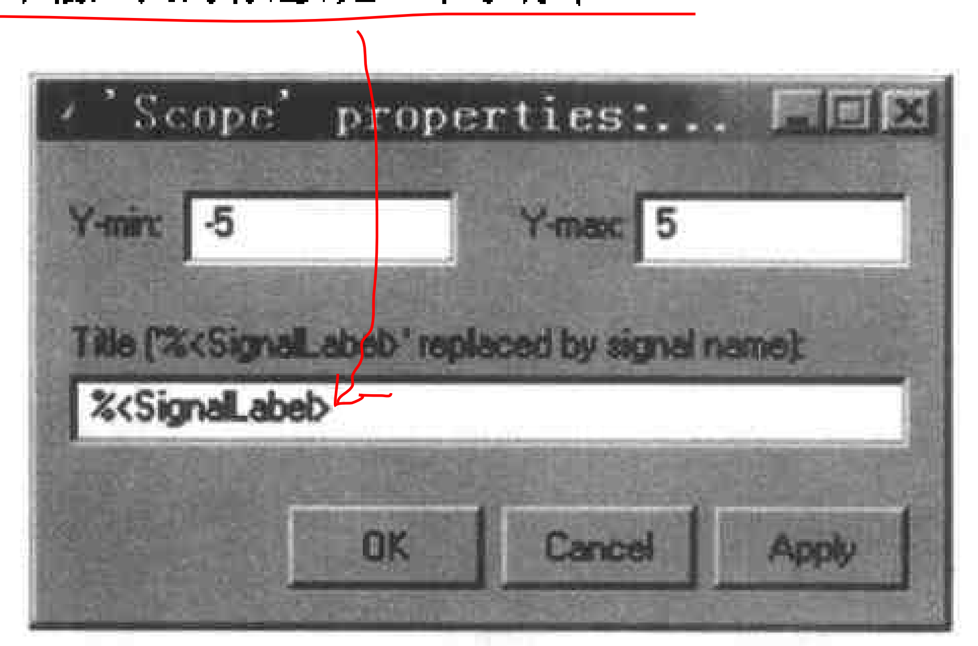


图 7.20　Y 轴极限设置对话框

3) 时间偏移量(Time Offset). 时间偏移量域显示在水平轴上与 0 相对应的时间. 要从坐标轴上得到确切的时间值范围,应当加上这一偏移量.

4) 自动调整 Scope 轴的刻度. 如果在仿真进行的时候点击 Auto-scale 按钮,坐标轴根据在当前屏幕中显示的数据自动调整刻度,并且自动刻度限被保存为缺省值. 这样就可以在另外的仿真中使用相同的界限.

5) 缩放数据(Zoom). 可以同时在横轴和纵轴方向或者仅在某一方向上缩放数据. 当

仿真正在运行时缩放特性不起作用.

要同时在纵横轴方向缩放数据,在选取工具条最左边的 Zoom 按钮后,使用一个边线框定义缩放的区域,在松开鼠标时,Scope 窗口显示这一区域的数据.

如果仅仅需要在横轴方向上缩放数据,点击工具条中间的 Zoom 按钮. 定义缩放区域时定位鼠标指针于区域的一端,然后按下鼠标键并拖动到区域的另一端松开鼠标. 在松开鼠标键时,Scope 显示被放大的区域.

在纵轴方向进行缩放时与在横轴方向进行缩放的做法相同,只是在定义缩放区域之前应该按最右边的 Zoom 工具按钮.

6) 保存坐标轴的设置. 通过 Save axes settings 工具按钮能够保存当前的横轴和纵轴的设置以便在另一个仿真时能够使用它们. 可能在对显示数据的一个区域进行缩放后想保存它的坐标设置以在另外的仿真中观察同一区域的数据.

7) Scope 属性(Properties). 通过选择 Scope 窗口工具条中的 Properties 按钮来改变坐标轴限,设置坐标轴数、时间范围、标记、采样参数以及保存选项. 按下工具条中的 Properties 按钮,则显示如图 7.21 所示的对话框.



图 7.21　Scope 属性设置窗口及 General 页面

该对话框窗口包含两个页面:General 和 Data history.

8) 一般(General)参数. 在 General 页面可以设置坐标轴参数、时间范围、标记,还可以在该页面选择浮动显示(floating scope)选项.

坐标轴数(Number of axes),在该数据域中设置 Y 轴数. 除了浮动显示外,Scope 模块包含的坐标轴数量没有限制. 所有坐标轴共用一个时间基准(X 轴),但具有独立的 Y 轴,坐标轴的个数等于输入端口的个数.

时间范围(Time range),通过输入一个数或 auto 到该域中来改变 X 轴限制. 输入一个秒数值,则每个窗口显示相应秒数的数据量;输入 auto,则设置 X 轴为仿真持续时间;该域中不能输入变量名.

刻度标记(Tick labels),可以选择刻度标记于所有坐标轴、一个坐标轴或底部坐标轴

上，也可不作刻度标记. 在 Tick labels 下拉框中选择：all，none，或 on the bottom axis only.

浮动显示(Floating scope)，浮动 Scope 是能够显示一条或多条信号线上的信号的 Scope 模块.

要在模型中加进一个浮动 Scope 模块，拷贝 Scope 模块到模型窗口，再打开该模块，选择显示窗口工具条上的 Properties 按钮，然后选取 General 页中的 Floating scope 核选框.

要在仿真期间使用浮动 Scope，首先打开该模块，要显示某条信号线上的信号，选取该条线，要选取多条线时在点击其它的线的同时按住 Shift 键. 有可能需要点击 Scope 工具条上的 Auto-scale data 按钮以找到信号并调整坐标轴到信号的值附近. 浮动 Scope 不能显示多个坐标轴.

模型中可以包含多个浮动 Scope 模块，一般来讲，在一个模型窗口中使用多个浮动模块没有多大用处，因为其显示内容是一样的.

还可以通过选择 Properties 工具按钮改变坐标界限.

抽样(Sampling)，在 General 页的抽样(Sampling)下拉菜单中可选择：Decimation 或 Sample time 选项，其缺省选项为：Decimation. 选择 Decimation 后，在其右边的数据域中输入抽取因子，其缺省值为 1；选择 Sample time 后，在其右边的数据域中输入采样时间，其缺省值为 0.

9）控制数据保存和显示. 可以通过设置 Data History 页面中相应的域控制 Scope 保存和显示数据的数量. 图 7.22 显示了 Data History 页面中的参数情况. 在该页面中还可以选择保存数据到工作空间. 要应用当前的参数和选项，按 OK 或 Apply 按钮. 这些参数值将在下次仿真生效.

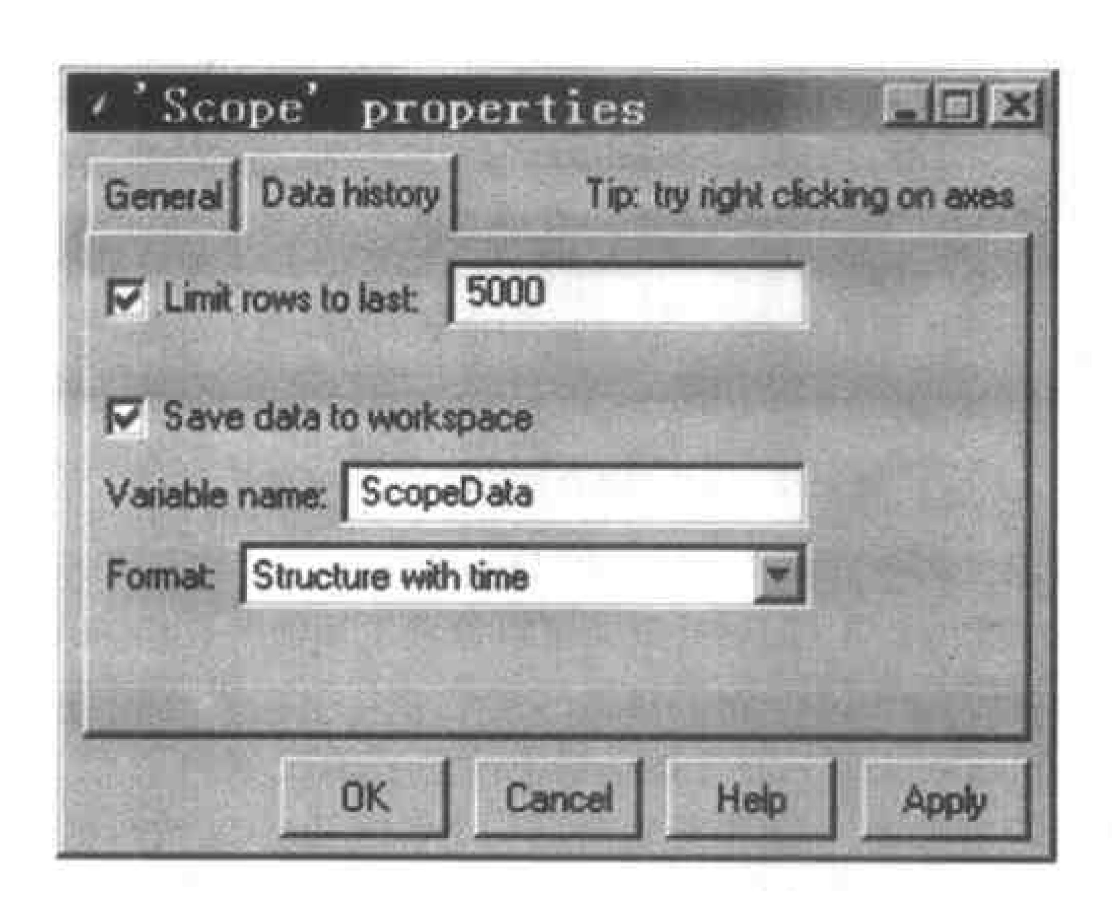


图 7.22 Scope 属性设置窗口的 Data History 页面

持续极限行(Limit rows to last)，在选择了 Limit rows to last 核选框后，可在其数据域中输入该参数值. Scope 依赖其数据历史记录来缩放或自动缩放. 如果极限行为 2000，而仿真产生 3000 行，则只有最后 2000 行可用于重新显示.

保存数据至工作空间(Save data to workspace),通过选择 Save data to workspace 核选框,在仿真结束,可以自动将 Scope 采集的数据保存. 选择 Save data to workspace 核选框后,Variable name(变量名)和 Format(格式)域就是活动的,可输入变量名和格式.

变量名(Variable name),在 Variable name 域中输入变量名,变量名必须是模型中使用的各种变量标识中惟一的. 其它数据标识变量有:其它 Scope 模块定义的,To Workspace 模块定义的,仿真返回变量(如时间、状态、输出). 让 Scope 保存至工作空间,就不必再将同样的数据流既送到 Scope block 模块又送到 To Workspace 模块.

格式(Format),数据可以保存的格式有三种:矩阵(Matrix),结构(Structure),具有时间的结构(Structure with time). 矩阵形式只用于具有一个坐标轴的 Scope,对于多坐标轴,要使用结构形式.

10) 打印 Scope 窗口的内容. 按 Scope 窗口工具条中最右边的打印按钮.

(3) 模块数据类型

该模块接受实数值信号,包括任意类型的同性质向量.

(4) 模块特点

1) 采样时间从驱动模块继承,也可以设置;

2) 状态 0.

7.2.3 Stop Simulation(停止仿真)

(1) 模块功能

当输入为非 0 值时停止仿真.

(2) 模块说明

当输入为非 0 值时 Stop Simulation 模块终止仿真.

仿真在终止之前完成当前时间步的计算. 如果该模块的输入是向量,任何非 0 的向量元素都会导致仿真停止. 可以使用该模块与 Relational Operator(逻辑运算)模块相连,以控制仿真什么时候停止. 例如,如图 7.23 所示的模型,当输入信号达到 9.1 时停止仿真.

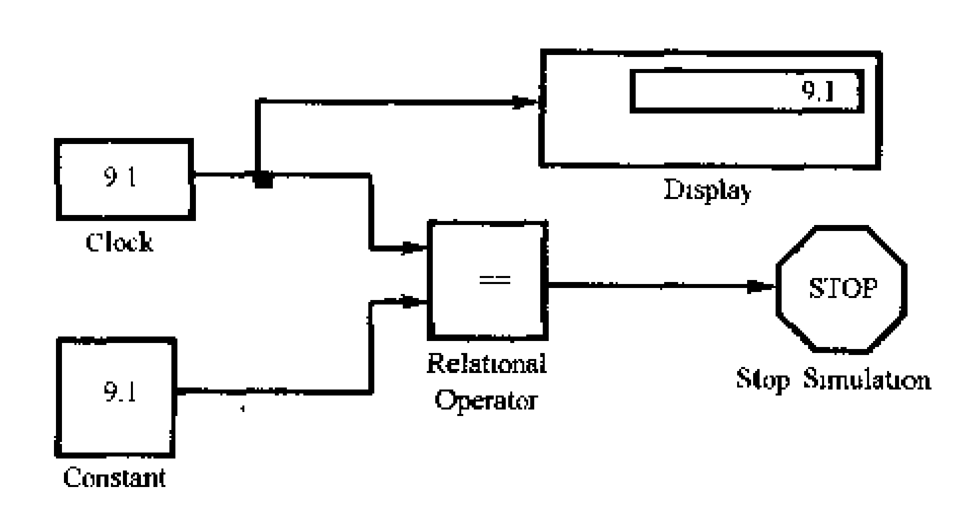


图 7.23 Stop Simulation 模块示例

(3) 模块数据类型

该模块接受双精度类型实数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.24 所示.

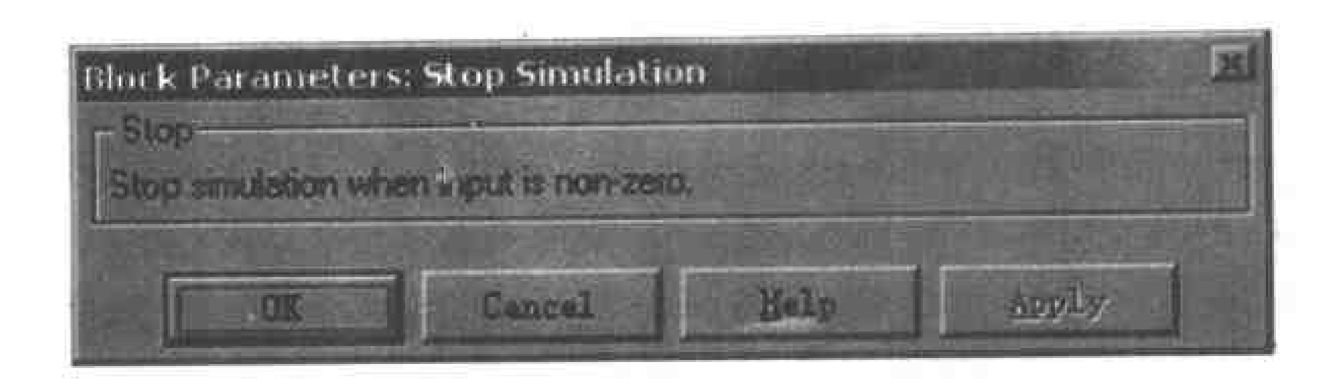


图 7.24 Stop Simulation 模块参数对话框

(5) 模块特点

可向量化.

7.2.4 To File(写入文件)

(1) 模块功能

将数据写到文件.

(2) 模块说明

To File 模块将其输入写到 MAT 文件中的矩阵. 该模块将每一时间步写成一列:第一行是仿真时间;该列中剩余的行是输入的数据,输入向量中每一元素占一数据点. 矩阵形式如下:

$$\begin{bmatrix} t_1 & t_2 & \cdots & t_4 \\ ul_1 & ul_2 & \cdots & ul_{\text{final}} \\ \vdots & \vdots & \ddots & \vdots \\ un_1 & un_2 & \cdots & un_{\text{final}} \end{bmatrix} \tag{7.7}$$

From File 模块能够毫无修改地使用 To File 模块写的数据. 然而,From Workspace 得到的矩阵是 To File 模块写入的矩阵的转置.

该模块边仿真边写数据,在仿真结束时数据写入完成. 模块的图标显示指定的输出文件的名字. 如果指定文件已存在,那么在仿真时将覆盖该文件.

写入数据的数量和在哪些时间步写入数据由模块的参数确定.

1) Decimation 参数,允许每 n 个采样写入一组数据,其中 n 是降采样因子. 缺省的降采样因子为 1,它使得每一时间步都写入数据.

2) Sample time 参数,可以指定数据采集的采样间隔. 这一参数在使用变步长求解器时是有用的,因为它的时间步之间的间隔可能不一样. 该参数的缺省值是 −1,它使得该模块在确定写入哪些点时从驱动它的模块那儿继承采样时间.

(3) 模块数据类型

该模块接受双精度类型实数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.25 所示.

1) 文件名(Filename),指定放矩阵的 MAT 文件的名称.

2) 变量名(Variable name),指文件中包含的矩阵的名字.

3) 抽样(Decimation),指抽样因子. 缺省值为 1.

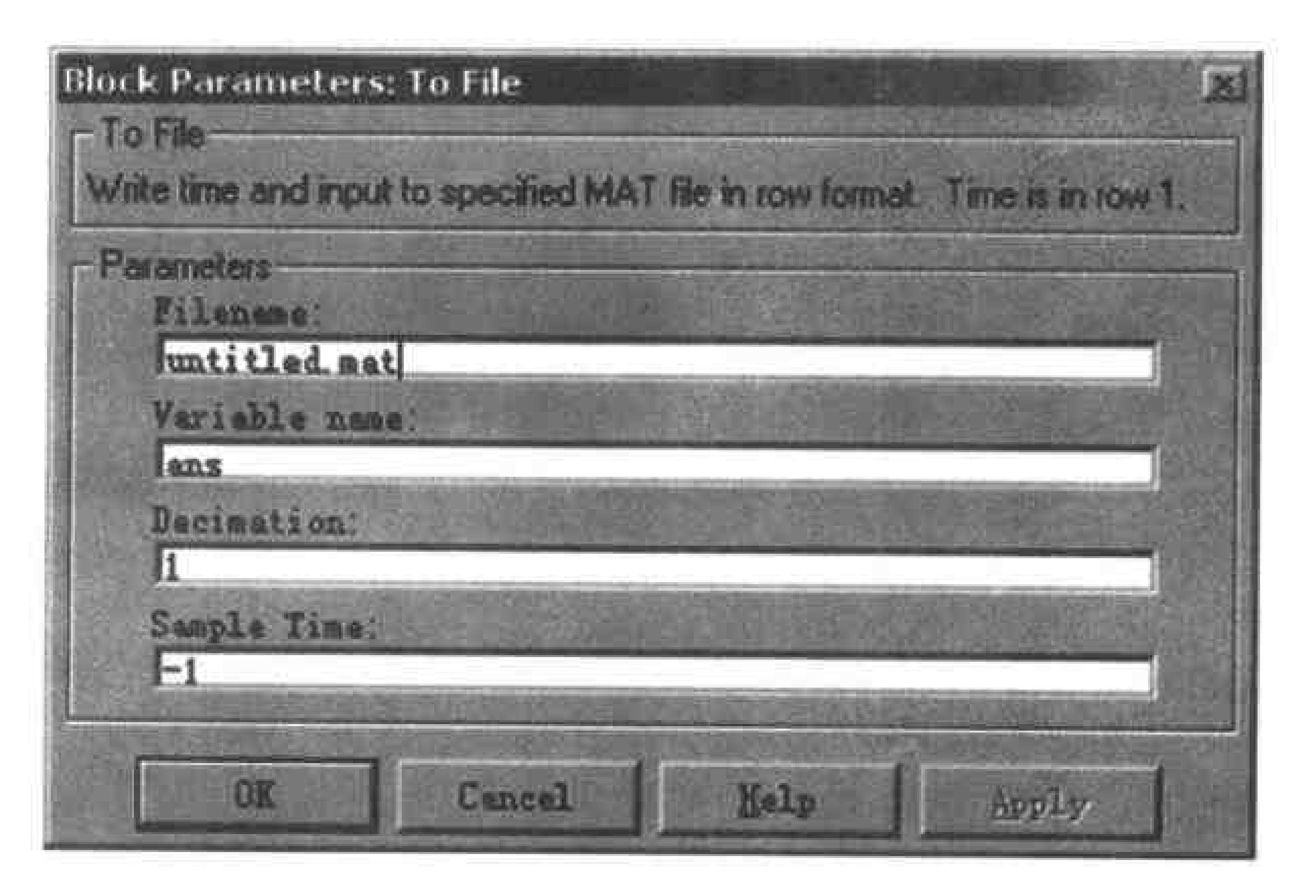


图 7.25 To File 模块参数对话框

4) 采样时间(Sample time),采集数据点的采样时间.

(5) 模块特点

1) 采样时间从驱动模块继承;

2) 可向量化.

7.2.5 To Workspace(写到工作空间)

(1) 模块功能

写数据到工作空间.

(2) 模块说明

To Workspace 模块写它的输入到工作空间中.该模块写其输出到由参数变量名(Variable name)指定的矩阵或结构中.参数保存格式(Save format)确定输出格式.

1)矩阵.矩阵具有如下的形式:

$$\begin{bmatrix} u1_1 & u2_1 & \cdots & un_1 \\ u1_2 & u2_2 & \cdots & un_2 \\ \vdots & \vdots & \ddots & \vdots \\ u1_{\text{final}} & u2_{\text{final}} & \cdots & un_{\text{final}} \end{bmatrix} \tag{7.8}$$

写入数据的数量,以及在哪些时间步写入数据由模块的参数确定:

Maximum number of rows 参数表明保存多少行数据.如果仿真产生的行数比指定的最大行数要多,仿真只保存最近产生的行.要保存所有数据,将该参数的值设为 inf.

Decimation 参数,允许每 n 个采样写入一组数据,其中 n 是降采样(抽样)因子.缺省的降采样因子为 1,它使得每一时间步都写入数据.

Sample time 参数,可以指定数据采集点的采样间隔.这一参数在使用变步长求解器时是有用的,因为它的时间步之间的间隔可能不一样.该参数的缺省值是-1,它使得该模块在确定写入哪些点时,从驱动它的模块那儿继承了采样时间.

在仿真期间,模块写数据到内部缓冲区,当仿真结束或者停止时,数据被写到工作空间.该模块的图标显示被写入数据的矩阵的名字.

2）结构. 这种格式的结构由三个字段组成：时间、信号和模块名. 时间字段为空；模块名字段包含 To Workspace 模块的名字；信号字段包含一个两字段的结构：值和标识. 值字段包含信号值矩阵.

3）具有时间的结构. 这一格式与结构格式相同，只是其时间字段包含仿真时间步向量.

4）使用 From File 模块保存的数据. 如果用 To Workspace 模块写的数据，随后将被 From File 模块保存或者读取，必须加进时间数据，而且矩阵应该转置.

5）用 From Workspace 模块保存的数据. 如果用 To Workspace 模块写的数据，将在另一个仿真时用 From Workspace 模块"回放"，必须加进仿真时间的值. 加进时间值的方法依赖于保存格式.

如果保存格式是结构，可以选择具有时间的结构来包含仿真时间作为保存格式，那么在输出结构中就保存有仿真时间向量.

如果保存格式是矩阵，必须加入一列仿真时间. 方法有两种：

将 Clock 模块的输出作为多路输入给 To Workspace 模块的向量输入线的第一个元素.

通过仿真参数(Simulation Parameters)对话框的返回值或从命令行指定时间. 当仿真结束时，你可以使用下面的命令将时间向量 t 与矩阵合并成新的矩阵：

```
matrix=[t; matrix]
```

（3）模块数据类型

该模块可以保存任何实数或复数类型的数据到 MATLAB 工作空间.

（4）模块参数对话框

该模块的参数对话框如图 7.26 所示.

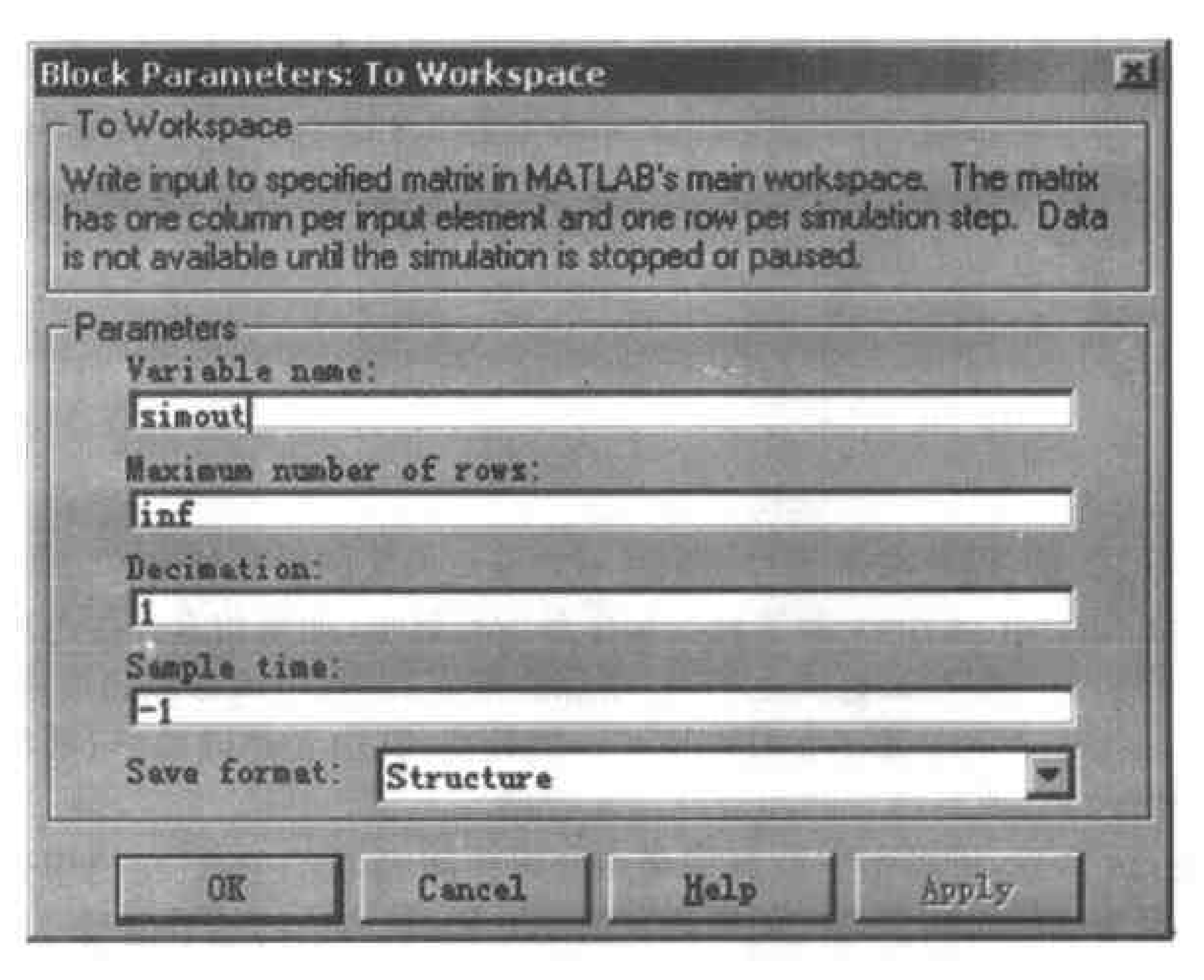


图 7.26 To Workspace 模块参数对话框

1）变量名(Variable name)，存放数据的矩阵名.

2) 最大行数(Maximum number of rows),存放数据的最大行数.缺省值为 1000 行.

3) 抽样(Decimation),指抽样因子.缺省值为 1.

4) 采样时间(Sample time),采集数据点的采样时间.

5) 保存格式(Save format),保存仿真输出到工作空间的格式,有三个选项:具有时间的结构,结构和矩阵.缺省为结构.

(5) 模块特点

1) 采样时间继承;

2) 可向量化.

7.2.6 XY Graph(显示平面图形)

(1) 模块功能

使用 MATLAB 的图形(figure)窗口显示信号的 X-Y 图.

(2) 模块说明

XY Graph 模块在 MATLAB 的图形窗口中显示它的输入信号的 X-Y 图.

该模块有两个标量输入.模块绘制第一个输入的数据(X 轴方向)对第二个输入的数据(Y 轴方向)的曲线图.该模块对于检验两状态的数据是有用的.超过指定范围的数据将不显示.

在仿真开始的时候,Simulink 为每个 XY Graph 模块打开一个图形窗口.

要运行说明 XY Graph 模块的用法的演示,在命令窗口输入 lorenzs.

(3) 模块数据类型

该模块接受双精度类型实数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.27 所示.

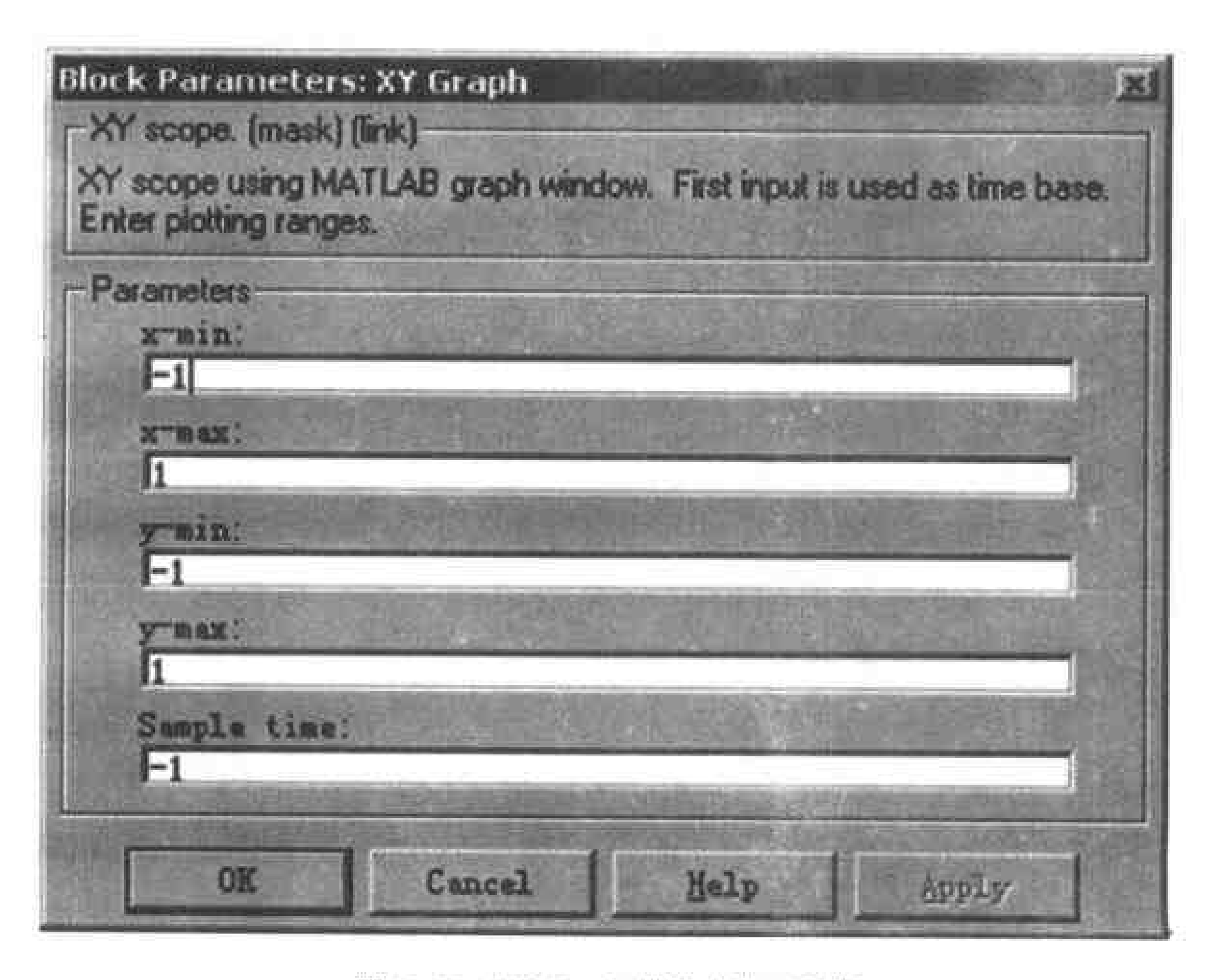


图 7.27 XY Graph 模块参数对话框

1) x min,x 轴最小值.缺省值为 −1.

2) x-max,x 轴最大值.缺省值为 1.

3) y-min,y 轴最小值.缺省值为 −1.

4) y-max,y 轴最大值.缺省值为 1.

5) 采样时间(Sample time),采样间隔时间.缺省为 −1,即采样时间由其驱动模块决定.

(5) 模块特点

1) 采样时间从驱动模块继承;

2) 状态 0.

7.3 Discrete 库中的模块

Discrete 库中包含的模块如表 7.4 所列,各个模块的图标如图 7.4 所示.

表 7.4 Discrete 库中的模块

模块名	功　能
Discrete Filter	实现 IIR 和 FIR 滤波器
Discrete State Space	实现用离散状态方程描述的系统
Discrete Time Integrator	执行信号的离散时间积分
Discrete Transfer Fcn	实现离散传递函数
Discrete Zero Pole	实现以零极点形式描述的离散传递函数
First Order Hold	实现一阶采样保持
Unit Delay	将信号延迟一个采样周期
Zero Order Hold	实现一个采样周期的零阶保持

7.3.1 Discrete Filter(离散滤波器)

(1) 模块功能

实现 IIR 或 FIR 滤波器.

(2) 模块说明

Discrete Filter 模块实现无限脉冲响应(IIR)和有限脉冲响应(FIR)滤波器.可以使用 Numerator 和 Denominator 参数以向量的形式指定分子和分母的 z^{-1}的升幂多项式的系数.分母的阶数必须大于或者等于分子的阶数.

Discrete Filter 模块表达了信号处理人员经常使用的方法,他们使用 z^{-1}(延迟算子)的多项式,描述数字滤波器.Discrete Transfer Fcn 模块代表控制工程人员经常使用的方法,他们使用 z 的多项式描述离散系统.当分子和分母的长度相等(阶数相等)时,两种方法是相同的.一个有 n 个元素的向量描述一个 $n-1$ 阶多项式.

根据指定的分子分母,在模块的图标中显示相应的分子分母表达式.

(3) 模块数据类型

该模块接受和输出双精度类型实数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.28 所示.

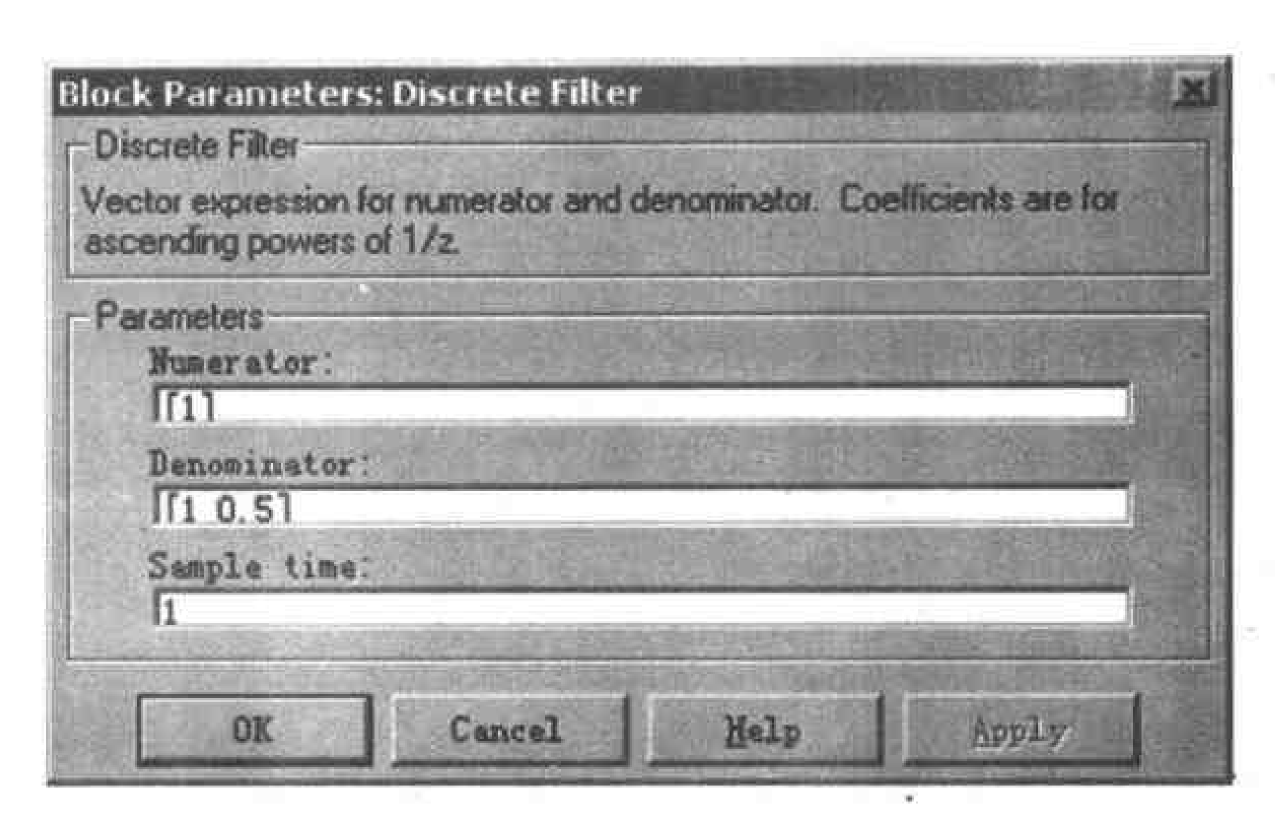


图 7.28 Discrete Filter 模块参数对话框

1) 分子(Numerator),指分子系数向量.缺省值为 1.

2) 分母(Denominator),指分母系数向量.缺省值为[1 0.5].

3) 采样时间(Sample time),采样间的时间间隔.

(5) 模块特点

1) 只有分子与分母参数长度相同,才直接馈通;

2) 采样时间是离散的;

3) 没有标量扩展;

4) 状态为分母长度−1;

5) 不可向量化;

6) 没有过零区间.

7.3.2 Discrete State-Space(离散状态空间)

(1) 模块功能

实现用离散的状态方程给出的系统.

(2) 模块说明

Discrete State-Space 模块实现由式(7.9)描述的系统.

$$
\begin{aligned}
x(n+1) &= Ax(n) + Bu(n) \\
y(n) &= Cx(n) + Du(n)
\end{aligned}
\tag{7.9}
$$

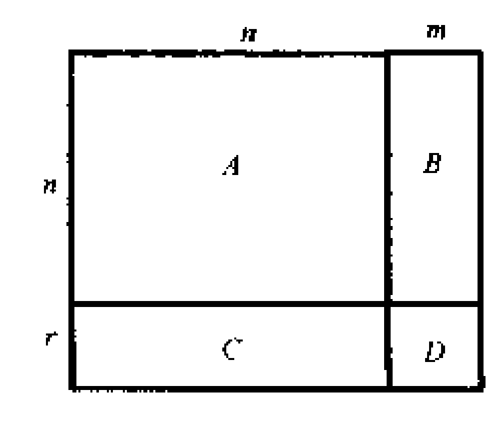


图 7.29 系数矩阵要求示意图

式中 u 是输入,x 是状态,y 是输出.系数矩阵 A,B,C,D 必须满足如图 7.29 所示的要求.

A 必须是一个 $n \times n$ 的矩阵,其中 n 是状态的个数.

B 必须是一个 $n \times m$ 的矩阵,其中 m 是输入的个数.

C 必须是一个 $r\times n$ 的矩阵，其中 r 是输出的个数.

D 必须是一个 $r\times m$ 的矩阵.

该模块接受一个输入并且产生一个输出. 输入向量的宽度由矩阵 B 和 D 的列数确定. 输出向量的宽度由矩阵 C 和 D 的行数确定.

Simulink 将包含有较多 0 的矩阵转换为稀疏矩阵以实现快速乘法.

(3) 模块数据类型

该模块接受和输出双精度类型实数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.30 所示.

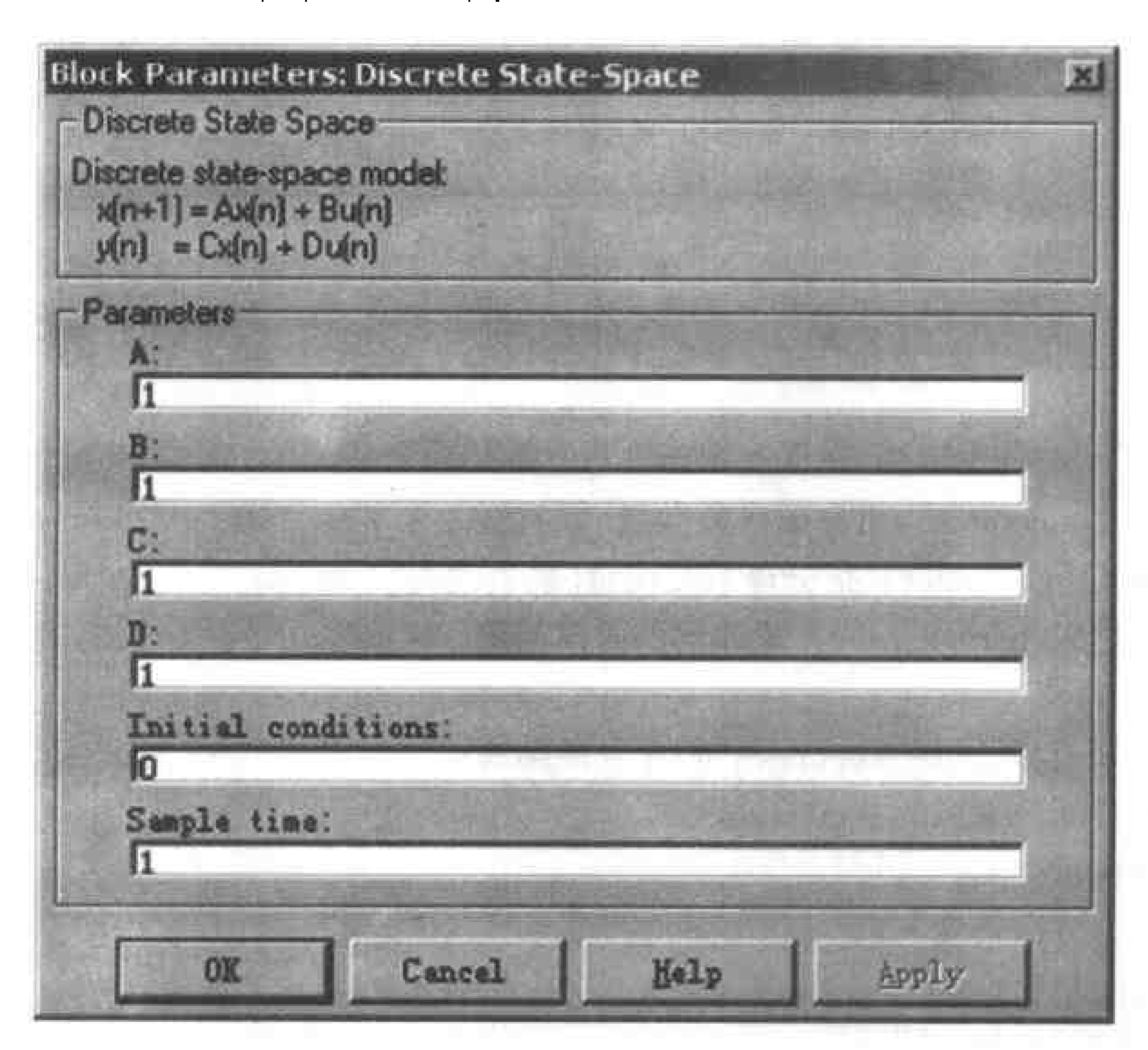


图 7.30 Discrete State-Space 模块的对话框

1) A,B,C,D:是由式(7.9)定义的系数矩阵.

2) 初始条件(Initial conditions)，指初始条件向量. 缺省值为 0.

3) 采样时间(Sample time)，采样时间间隔.

(5) 模块特点

1) 只有 D 不等于 0，才直接馈通；

2) 采样时间是离散的；

3) 初始条件有标量扩展；

4) 状态由 A 的大小决定；

5) 可向量化；

6) 没有过零区间.

7.3.3　Discrete-Time Integrator(离散时间积分器)

(1) 模块功能

实现一个信号的离散时间积分.

(2) 模块说明

在构造一个纯离散系统时,可以用 Discrete-Time Integrator 模块代替 Integrator 模块.

通过 Discrete Time Integrator 模块可以在模块的对话框中或者模块的输入中定义初始状态;输出模块状态;定义积分的上下限;根据另外的一个复位输入对状态复位.

1) 积分方法.该模块使用的积分方法有:前向欧拉法、后向欧拉法和梯形法.对于给定的积分步 k,Simulink 更新 y(k)和 x(k+1).T(触发情况下采样时间 ΔT)是采样周期.值可以根据上下限进行删减.所有情况下 x0=IC.

前向欧拉法(缺省方法),也叫前向矩形法或者左手近似.对于这种方法,用 T/(z-1)来近似 1/s.这样得到 y(k)=y(k-1)+T * u(k-1).

让 x(k)=y(k)得到

x(k+1)=x(k)+T * u(k)

y(k)=x(k)

使用此方法,输入端口 1 没有直接馈通.

后向欧拉法,也叫后向矩形法或者右手近似.对于这种方法,用 T * z/(z-1)来近似 1/s.这样得到 y(k)=y(k-1)+T * u(k)

让 x(k)=y(k-1),得到

x(k+1)=y(k)

y(k)=x(k)+T * u(k)

此方法,输入端口 1 具有直接馈通.

梯形法.对于这种方法,用 T/2 * (z+1)/(z-1)近似 1/s.这样得到

$$y(k) = y(k-1) + T/2 * (u(k) + u(k-1))$$

当 T 固定时(等于采样周期),让 x(k)=y(k-1)+T/2 * u(k-1),得到

$$x(k+1) = y(k) + T/2 * u(k)$$
$$y(k) = x(k) + T/2 * u(k)$$

式中 x(k+1)是下一个输出的最佳估计.在 x(k)!=y(k)的意义上,它不正好是状态.

当 T 可变时(也就是说,它由触发时间得到),得到

$$x(k+1) = y(k)$$
$$y(k) = x(k) + T/2 * (u(k) + u(k-1))$$

此方法,输入端口 1 具有直接馈通.

该模块的图标显示了选择的积分方法,如图 7.31 所示.

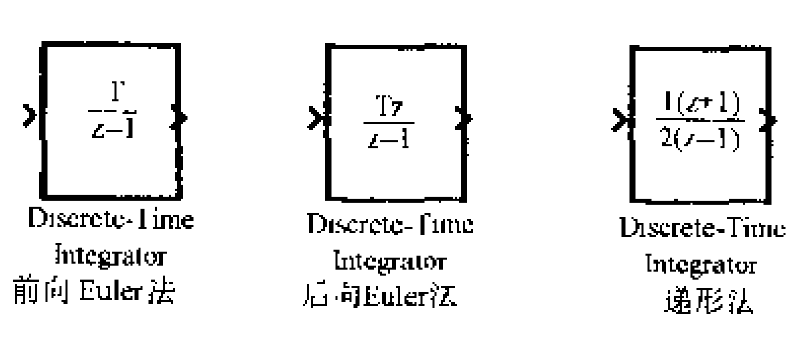


图 7.31　三种离散时间积分器模块的图标

2)定义初始条件.可以在模块的对话框中以一个参数的形式或者以一个外部信号输入的形式定义模块的初始条件:

要以模块参数的形式定义初始条件,指

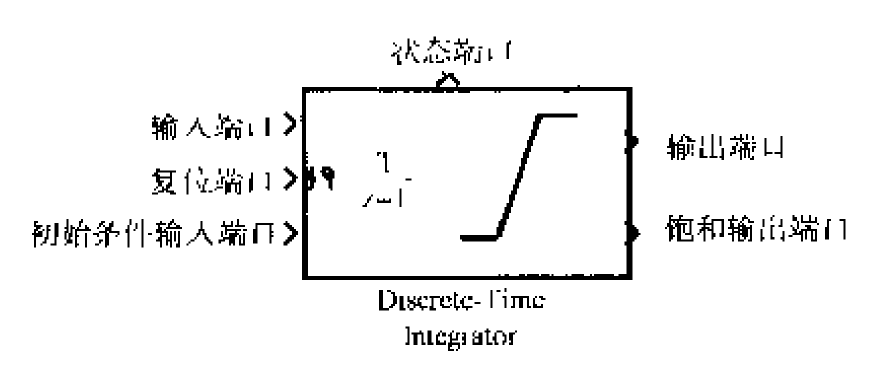


图 7.32 离散时间积分器模块在图标中显示端口

定 Initial condition source 参数为 internal，并且在 Initial condition 参数域中输入值.

要从一个外部信号源提供初始状态，指定 Initial condition source 参数为 external. 这时在模块输入端口的下部出现另外一个输入端口，如图 7.32 所示.

3)使用状态端口. 在两种已知情况下，必须使用状态端口而不是输出端口：

当模块的输出通过复位端口或者初始状态端口反馈给模块，形成了一个反馈回路.

如果想将状态从一个条件执行的子系统传给另外一个时，这时会出现时间上的问题.

要解决这些问题，可以将状态通过状态端口而不是输出端口传送. 尽管值相同，Simulink 产生它们的时间有细微的差别，这就保证了模型不会出现这些问题. 通过选择 Show state port 核选框，可以输出模块的状态.

缺省情况下，状态端口显示在模块的上端，如图 7.32 所示.

4）限制积分. 为了防止输出超出可指定的范围，选择 Limit output 核选框并在合适的参数域中输入限制参数. 这样做使得该模块在功能上像一个有限积分器. 当输出超出了限制时，积分过程被关断以防止积分结束. 在仿真期间，可以改变这些限制参数但不能改变输出是否是有限的. 输出由下列因素确定：

当积分结果小于 Lower saturation limit（下饱和限），并且输入为负时，输出保持在 Lower saturation limit.

当积分结果介于 Lower saturation limit（下饱和限）和 Upper saturation limit（上饱和限）之间时，输出积分结果.

当积分结果大于 Upper saturation limit（上饱和限），并且输入是正数时，输出保持在上饱和限.

要生成一个信号以能够表明其状态什么时候是受限的，选择 Show saturation port（显示饱和端口）核选框. 这时一个饱和端口显示在模块输出端口的底端，如图 7.32 所示.

饱和端口的信号取以下三个值之一：

1 ：表示积分超出了饱和上限；

0 ：表示积分没有超限；

−1：表示积分超出了饱和下限.

当选择了 Limit output 选项，模块有三个过零区间：一个检测它是何时达到了饱和上限；一个检测它是何时达到了饱和下限；另外一个检测它是何时离开饱和状态的.

5）复位状态. 模块可以通过一个外部信号复位状态为指定的初始条件. 要使模块重置它的状态，选择 External reset 选项. 一个触发端口显示在模块输入端口的下方并且显示了其触发类型，如图 7.32 所示.

当触发信号有上升沿时，选择 rising 触发状态复位.

当触发信号有下降沿时，选择 falling 触发状态复位.

当触发信号既有上升沿又有下降沿时，选择 either 触发状态复位.

复位端口是直接馈通的. 如果模块的输出，直接或通过一系列模块直接馈通反馈给了

这一端口，将会导致代数回路. 要解除这一回路，将状态端口传给复位端口. 要获得模块的状态，选择 Show state port 核选框.

6）选择所有选项. 选择所有选项后，模块图标如图 7.32 所示.

（3）模块数据类型

该模块接受和输出双精度类型实数信号.

（4）模块参数对话框

该模块的参数对话框如图 7.33 所示.

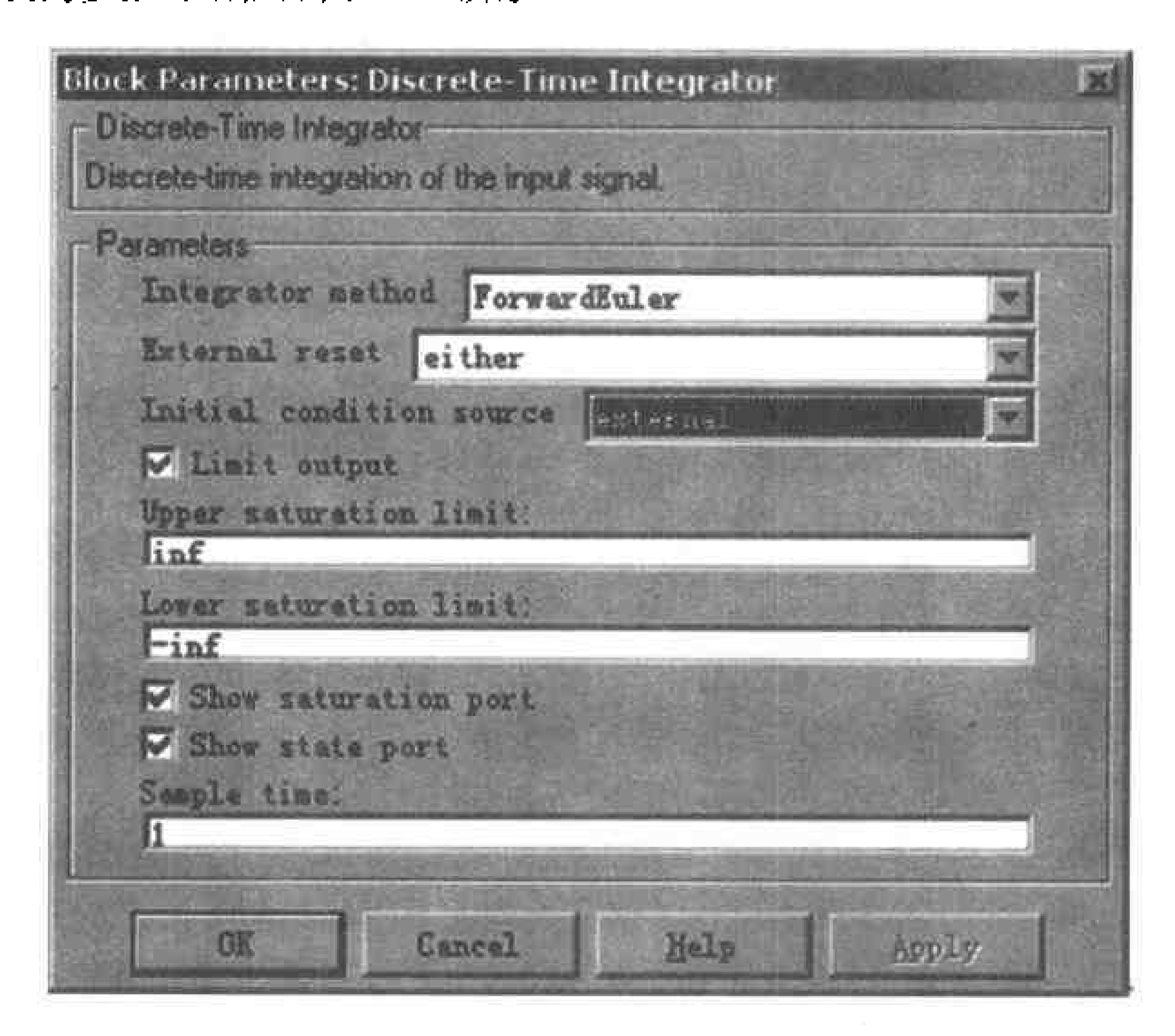


图 7.33 Discrete-Time Integrator 模块参数对话框

1）积分器方法(Integrator method)，指积分方法. 缺省值为：ForwardEuler.

2）外部复位(External reset)，当复位信号触发事件发生，复位其状态为初始条件.

3）初始条件源(Initial condition source)，指定获得状态初始条件是来自 Initial condition 参数(设为：internal)，还是外部模块(设为：external).

4）初始条件(Initial condition)，在初始条件源选 internal(内部)时，指状态初始条件.

5）限制输出(Limit output)，如果选择该核选框，将限制输出状态值在饱和下限和饱和上限参数之间.

6）饱和上限(Upper saturation limit)，指积分上限. 缺省为 inf.

7）饱和下限(Lower saturation limit)，指积分下限. 缺省为-inf.

8）显示饱和端口(Show saturation port)，如果选择该核选框，将在模块上加一个饱和输出端口.

9）显示状态端口(Show state port)，如果选择该核选框，将在模块上加一个状态输出端口.

10）采样时间(Sample time)，采样时间间隔. 缺省值为 1.

(5) 模块特点

1）复位和外部初始条件源端口，有直接馈通；

2）采样时间是离散的；

3）参数有标量扩展；

4）状态从驱动模块和参数继承；

5）可向量化；

6）检测复位有一个过零区间；检测上、下饱和限各有一个过零区间，离开饱和有一个过零区间.

7.3.4 Discrete Transfer Fcn(离散传递函数)

(1) 模块功能

实现离散传递函数.

(2) 模块说明

Discrete Transfer Fcn 模块实现用式(7.10)描述的 z 变换传递函数.

$$H(z)=\frac{mun(z)}{den(z)}=\frac{num_0z^n+num_1z^{n-1}+\cdots+unm_mz^{n-m}}{den_0z^n+den_1z^{n-1}+\cdots+den_n} \tag{7.10}$$

式中 $m+1$ 和 $n+1$ 分别是分子和分母系数的个数. *num* 和 *den* 按 z 的降幂包含有分子和分母的系数. *num* 可以是向量也可以是矩阵，但 *den* 必须是向量，可以在模块的对话框中指定这些参数. 分母的阶数必须大于等于分子的阶数.

模块的输入是标量，输出宽度等于分子的行数.

Discrete Transfer Fcn 模块代表了控制工程人员常用的典型方法，他们用 z 的多项式描述离散系统. Discrete Filter 模块代表了信号处理人员用的典型方法，他们用 z^{-1}(延迟因于)的多项式描述数字滤波器. 当分子分母的阶数相等时，这两种方法相同.

Discrete Transfer Fcn 模块在它的图标中显示了它是如何被指定的.

(3) 模块数据类型

该模块接受和输出双精度类型实数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.34 所示.

1）分子(Numerator)，为分子系数行向量. 一个具有多行数据的矩阵可产生多个输出. 缺省值为[1].

2）分母(Denominator)，分母系数行向量. 缺省值为[1　0.5].

3）采样时间(Sample time)，采样时间间隔. 缺省值为 1.

(5) 模块特点

1）只有当分子与分母参数长度相等时，才有直接馈通；

2）采样时间是离散的；

3）没有标量扩展；

4）状态为分母长度－1；

5）不可向量化；

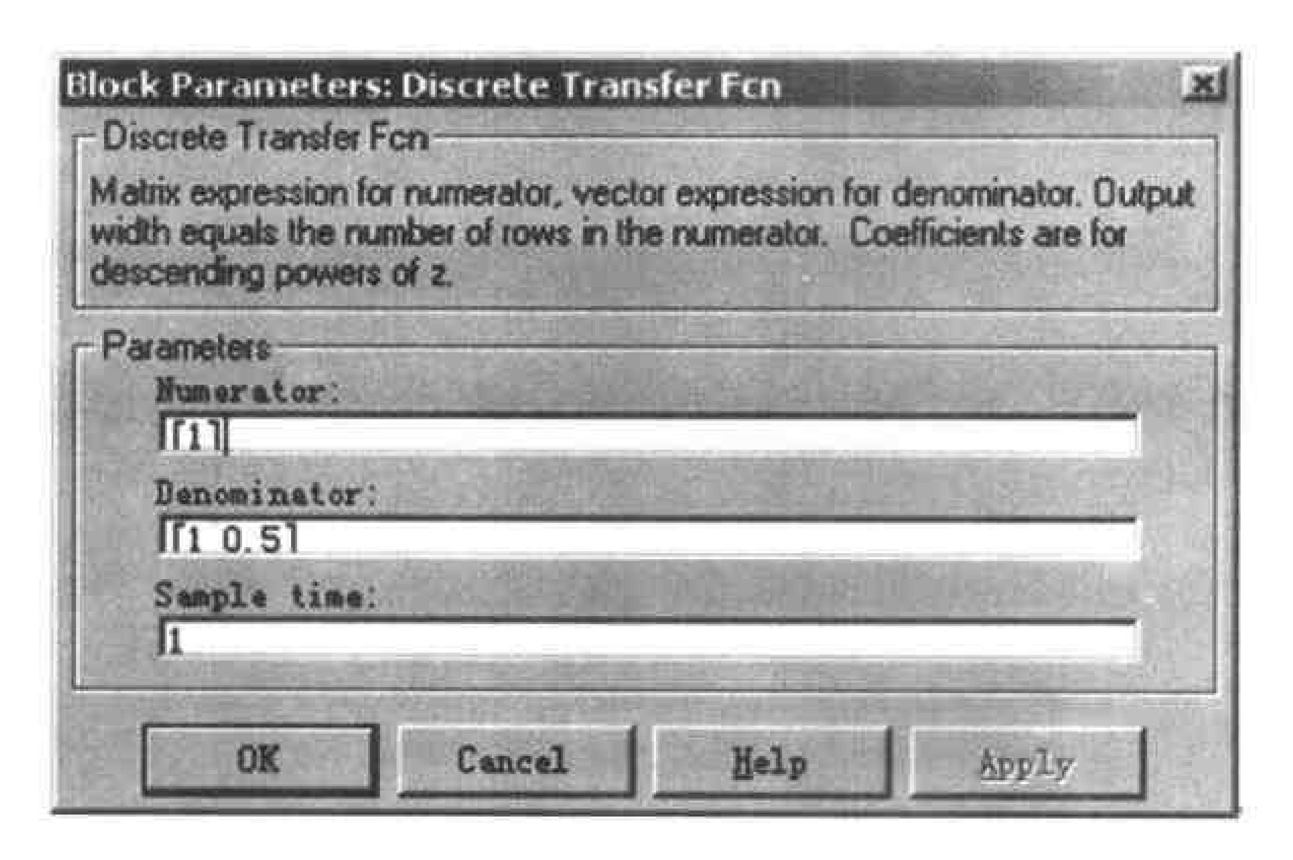


图 7.34 Discrete Transfer Fcn 模块参数对话框

6）没有过零区间.

7.3.5 Discrete Zero-Pole（数字零极点函数）

（1）模块功能

实现一个用零极点的形式指定的离散传递函数.

（2）模块说明

Discrete Zero-Pole 模块实现一个用延迟因子 z 的零点、极点和增益的形式给出的离散系统. 一个传递函数可以用零-极-增益的形式表达，对于 MATLAB 中的一个单输入单输出的系统，它们之间的关系可用式(7.11)表达.

$$H(z)=K\frac{Z(z)}{P(z)}=K\frac{(z-Z_1)(z-Z_2)\cdots(z-Z_m)}{(z-P_1)(z-P_2)\cdots(z-P_n)} \tag{7.11}$$

式中 Z 表示零点向量，P 表示极点向量，K 表示增益. 极点的数目必须大于或等于零点的数目($n\geqslant m$). 如果零点和极点是复数，它们必须是共轭复数对.

模块的图标显示了传递函数的指定参数.

（3）模块数据类型

该模块接受和输出双精度类型实数信号.

（4）模块参数对话框

该模块的参数对话框如图 7.35 所示.

1）零点(Zeros)，零点矩阵. 缺省值为 1.

2）极点(Poles)，极点向量. 缺省值为[0 0.5].

3）增益(Gain)，增益. 缺省值为 1.

4）采样时间(Sample time)，采样间时间间隔. 缺省值为 1.

（5）模块特点

1）如果零点与极点数相等时，有直接馈通；

2）采样时间是离散的；

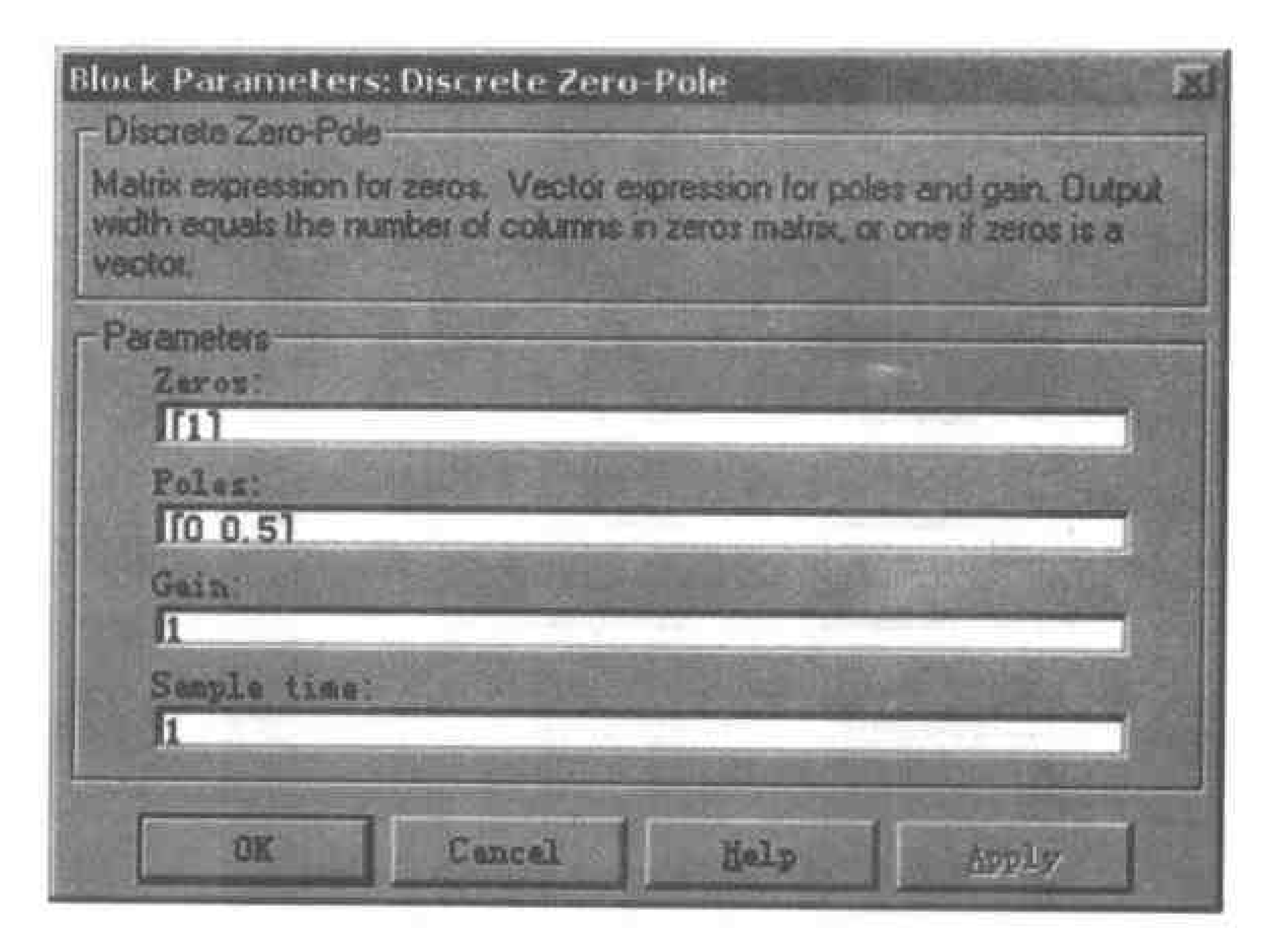


图 7.35 Discrete Zero-Pole 模块参数对话框

3）没有标量扩展；

4）状态为极点向量长度；

5）不可向量化；

6）没有过零区间.

7.3.6 First-Order Hold(一阶保持)

(1) 模块功能

实现第一级采样保持.

(2) 模块说明

First-Order Hold 模块实现以一定的采样间隔执行的一阶采样保持. 该模块在实际应用中没有多少价值,它主要用于学术研究.

可以通过运行演示程序 fohdemo 来观察零阶保持和一阶保持之间的区别. 图 7.36 比较了从 Sine Wave 模块和从 First-Order Hold 模块的输出结果.

(3) 模块数据类型

该模块接受和输出双精度类型信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.37 所示.

采样时间(Sample time),采样间的时间间隔. 缺省值为 1.

(5) 模块特点

1）没有直接馈通；

2）采样时间是连续的；

3）没有标量扩展；

4）状态为每个输入元素 1 连续和 1 离散；

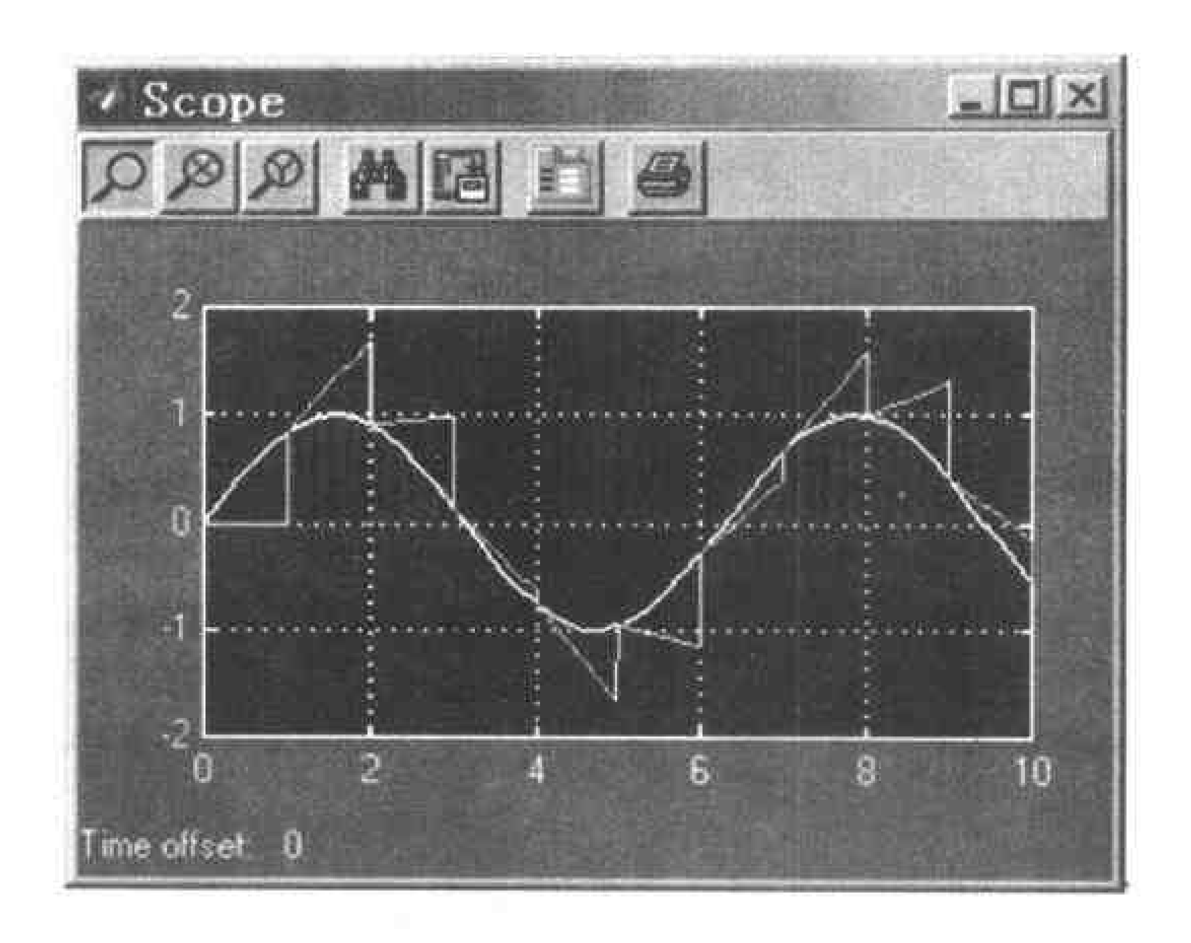


图 7.36 Sine Wave 模块和 First-Order Hold 模块输出结果比较

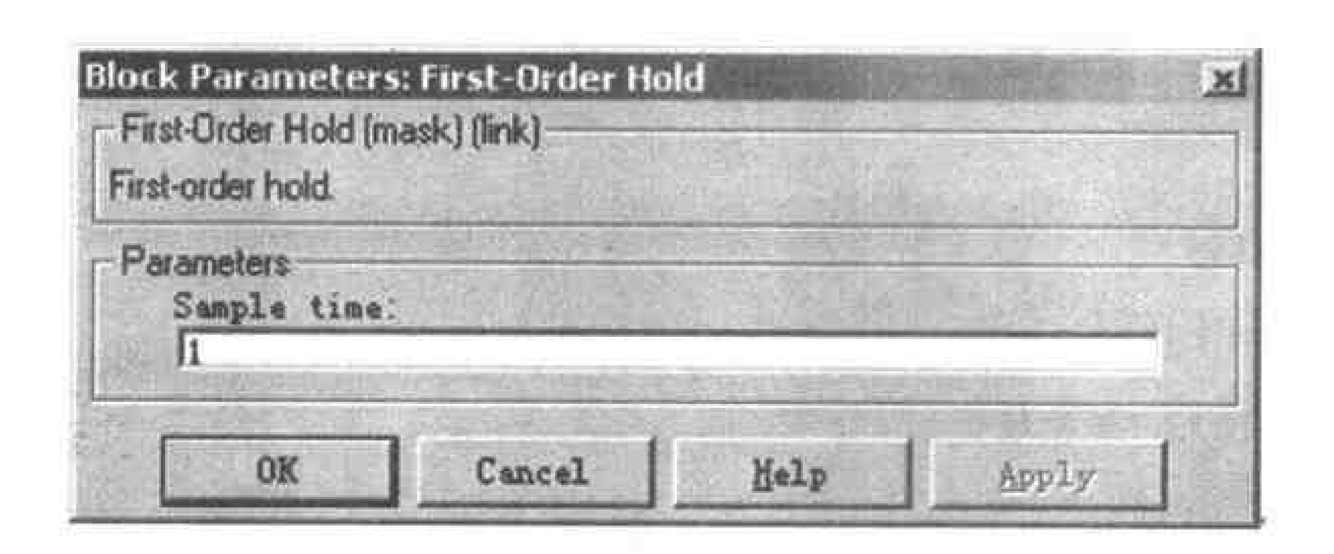


图 7.37 First-Order Hold 模块参数对话框

5）可向量化；

6）没有过零区间.

7.3.7 Zero-Order Hold（零阶保持）

（1）模块功能

实现一个采样周期的零阶保持.

（2）模块说明

Zero-Order Hold 模块实现指定采样率的采样和保持功能. 该模块有一个输入和一个输出，输入和输出可以是标量也可以是向量.

该模块可用于对一个或多个信号进行离散化或者以另外的速率对信号进行重新采样. 如果需要模拟采样，但又不需要另外的更为复杂的离散功能模块时，可以使用这一模块. 例如，用它连接 Quantizer 模块以模拟对输入有放大作用的 A/D 转换器.

（3）模块数据类型

该模块接受和输出任何类型的实数或复数值信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.38 所示.

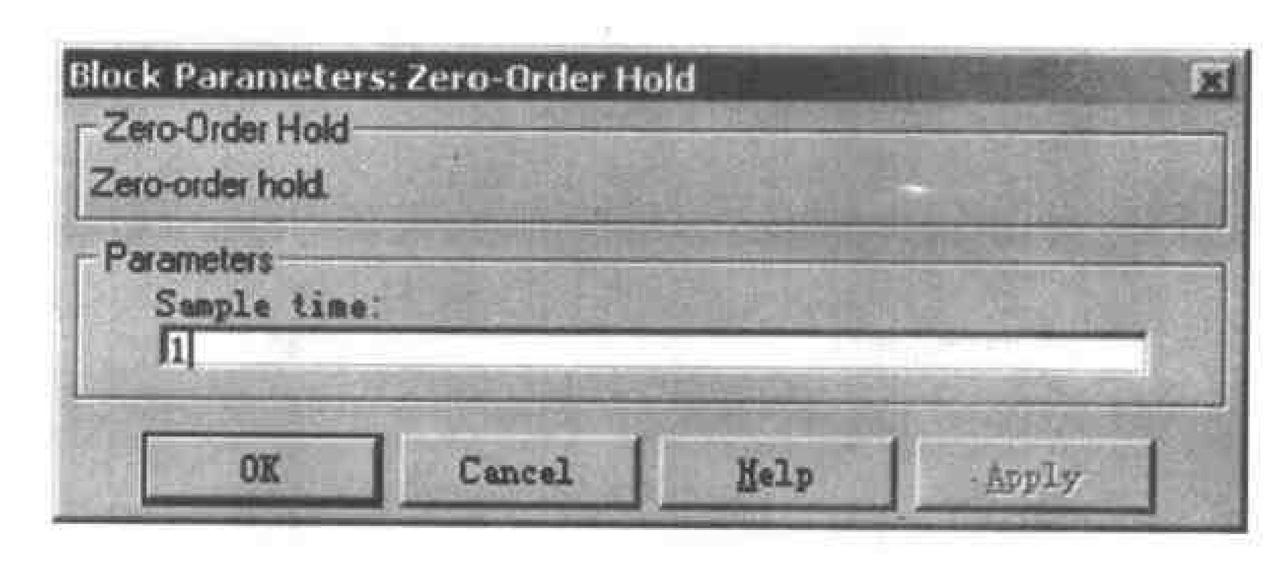


图 7.38 Zero-Order Hold 模块参数对话框

采样时间,采样间的时间间隔. 缺省值为 1.

(5) 模块特点

1) 有直接馈通;

2) 采样时间是离散的;

3) 有标量扩展;

4) 状态 0;

5) 可向量化;

6) 没有过零区间.

7.3.8 Unit Delay(单位延迟)

(1) 模块功能

将信号延迟一个采样周期.

(2) 模块说明

Unit Delay 模块将它的输入信号延迟并保持一个采样间隔. 如果模块的输入是向量,向量中所有元素的延迟时间都相同. 该模块与离散时间算子 z^{-1}的作用等价.

如果需要无延迟的采样保持函数,使用零阶保持(Zero-Order Hold)模块,或者如果需要大于一个单位的延迟,使用离散传递函数(Discrete Transfer Fcn)模块.

(3) 模块数据类型

该模块接受和输出任何类型的实数和复数信号,包括用户自定义数据类型. 对于用户自定义类型,初始条件必须为 0.

(4) 模块参数对话框

该模块的参数对话框如图 7.39 所示.

1) 初始条件(Initial condition),第一个仿真周期的模块输出. 该参数要仔细选取. 缺省值为 0.

2) 采样时间即采样间的时间间隔. 缺省值为 1.

(5) 模块特点

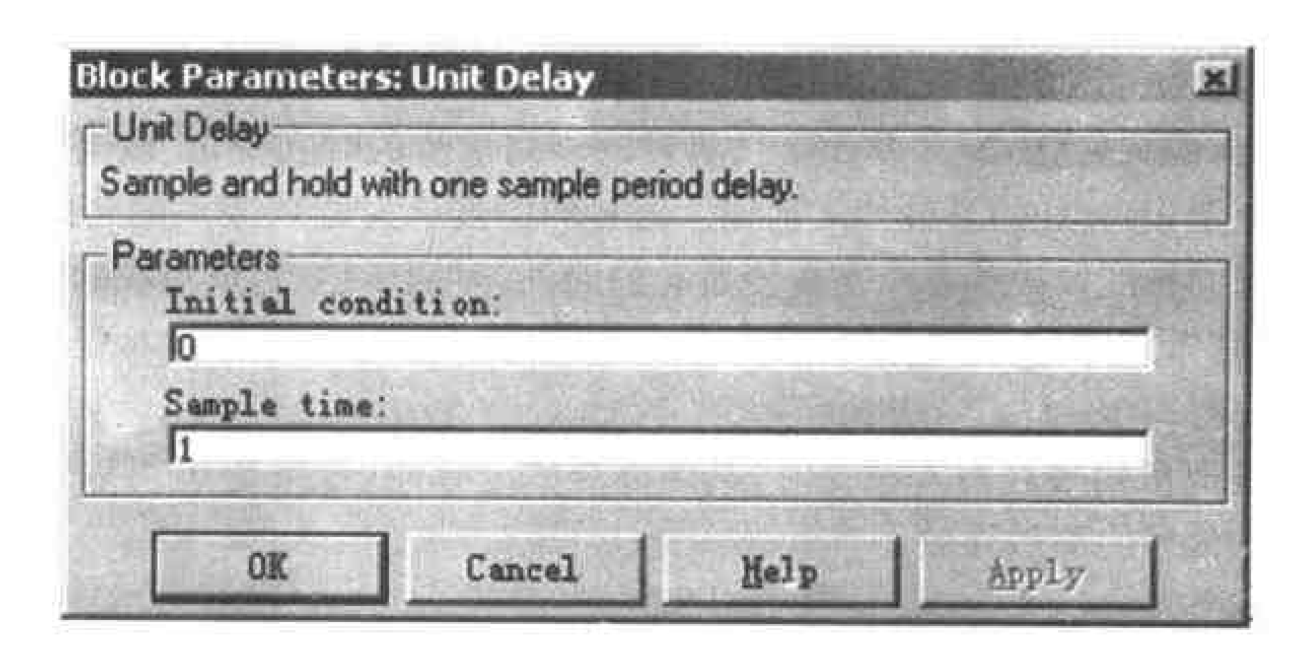


图 7.39 Unit Delay 模块的对话框

1）没有直接馈通；

2）采样时间是离散的；

3）初始条件参数或输入有标量扩展；

4）状态由驱动模块或参数继承；

5）可向量化；

6）没有过零区间.

7.4 Continuous 库中的模块

Continuous(连续)库中包含的模块如表 7.5 所列，各个模块的图标如图 6.3 所示.

表 7.5 Continuous 库中的各模块

模 块 名	功 能
Derivative	输入对时间的导数
Integrator	对信号进行积分
Memory	从前一时间步输出模块的输入
State-Space	实现线性状态空间系统
Transfer Fcn	实现线性传递函数
Transfer Delay	以给定的时间量延迟输入
Variable Transfer Delay	以可变的时间量延迟输入
Zero-Pole	实现用零极点形式表示的传递函数

7.4.1 Derivative(导数)

(1) 模块功能

输出输入对时间的导数

(2) 模块说明

Derivative 模块近似地给出其输入的导数，计算公式为

$$\frac{\Delta u}{\Delta t} \tag{7.12}$$

式中 Δu 是输入值的变化，Δt 是从上一仿真时间步到该步的时间变化. 该模块接受一个输入并且产生一个输出. 在仿真开始之前输入信号的值被认为是 0. 模块的初始输出是 0.

输出结果的精度取决于各个仿真步所花的时间. 时间步越短，该模块的输出曲线将会越光滑，结果越精确. 与有连续状态的模块不同，该模块的求解器在输入变化比较快时不会使用较短的时间步.

当输入是离散信号且输入改变时，其连续导数为脉冲，否则是 0. 可以使用式(7.13)得到一个离散信号的离散导数：

$$y(k) = \frac{1}{\Delta t}(u(k) - u(k-1)) \tag{7.13}$$

将式(7.13)作 Z 变换，得到

$$\frac{Y(z)}{U(z)} = \frac{1 - z^{-1}}{\Delta t} = \frac{z - 1}{z \cdot \Delta t} \tag{7.14}$$

使用 linmod 函数给一个包含有 Derivative 模块的模型进行线性化，将会遇到麻烦.

(3) 模块数据类型

该模块接受和输出双精度类型实数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.40 所示.

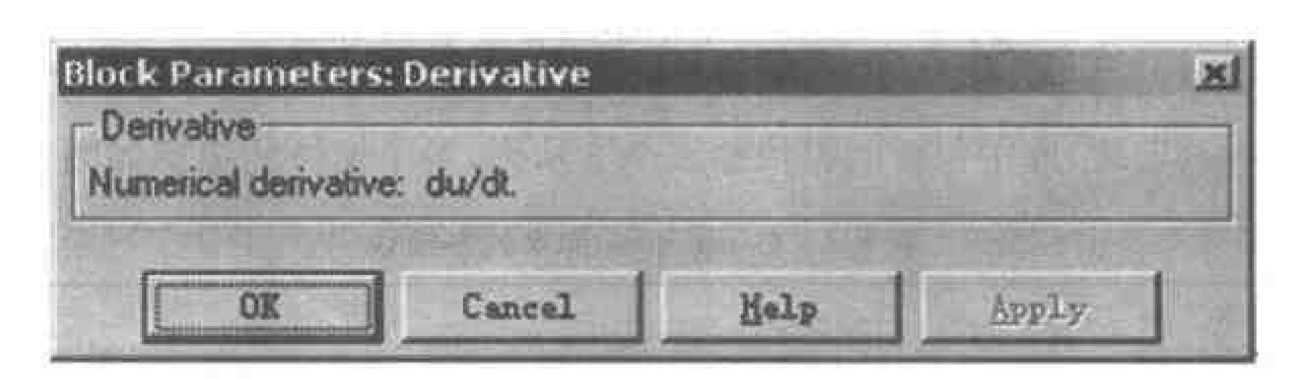


图 7.40 Derivative 模块参数对话框

(5) 模块特点

1) 有直接馈通；

2) 采样时间是连续的；

3) 标量扩展 N/A；

4) 状态为 0；

5) 可向量化；

6) 没有过零区间.

7.4.2 Integrator(积分器)

(1) 模块功能

对信号进行积分.

(2) 模块说明

Integrator 模块对其输入进行积分. 积分器的输出仅仅是其状态(积分). 通过 Integra-

tor 模块可以在模块的对话框中或作为模块的输入定义其初始状态;输出模块状态;定义积分结果的上下限;通过另外的一个复位输入对状态复位.

当构造一个纯离散系统时,应该用离散时间系统(Discrete-Time Integrator)模块.

1)定义初始状态.可以通过模块的对话框中的参数或者从外部输入一个信号来定义模块的初始状态:

以模块的参数的形式定义初始状态;指定 internal 作为 Initial condition source 参数的值,并且在 Initial condition 参数域中输入数值.

以一个外部信号源的形式提供初始状态;指定 external 作为 Initial condition sourcec 参数的值,在模块输入的下部会显示另外一个输入端口 — 初始条件输入端口.

2)使用状态端口.在两种已知的情况下,必须使用状态端口而不是输出端口:

当模块的输出通过复位端口或者初始状态端口反馈给模块,这将导致一个代数循环.

如果想将状态从一个条件执行的子系统传给另一个条件执行子系统时,则会导致时间问题.

可以通过将状态从状态端口而不是输出端口传出来纠正这些问题.尽管数值是相同的,Simulink 产生它们的时间稍微有所不同,这就保证了系统不会出现这些问题.可通过选取 Show state port 核选框输出模块的状态.缺省情况下,状态端口显示在模块的顶部.

3)限制积分结果。为了限制输出超出指定的范围,选择 Limit output 核选框,并在适当的参数域中输入限制参数.这样就使得该模块在功能上像一个有限积分器.当输出超出了限制时,积分过程被关断以防止积分结束.在仿真期间,可以改变这些限制参数但不能改变输出是否是有限的.输出由下列因素确定:

当积分结果小于饱和下限(Lower saturation limit),并且输入是负数时,输出保持在 Lower saturation limit.

当积分结果介于饱和下限(Lower saturation limit)和饱和上限(Upper saturation limit)之间时,输出是积分结果.

当积分结果大于饱和上限(Upper saturation limit),并且输入是正数时,输出保持在 Upper saturation limit.

要生成一个信号以能够表明其状态什么时候是受限的,选择显示饱和端口(Show saturation port)核选框.一个饱和端口显示在模块输出端口的底下.

饱和端口的信号取值有三个:

1 :表示积分超出了上限.

0 :表示积分没有超限.

-1:表示积分超出了下限.

如果选择了 Limit output 选项,模块有三个过零点检测:一个检测它是什么时候达到了饱和上限;一个检测它是什么时候达到了饱和下限;另外一个检测它是什么时候离开饱和状态.

4)复位状态.模块可以通过一个外部信号复位状态为指定的初始值.要使模块复位它的状态,选取 External reset 选项.一个触发端口显示在模块输入端口的下方,并显示了其触发类型:

当触发信号有上升沿时,选择 rising 触发状态复位.

当触发信号有下降沿时，选择 falling 触发状态复位.

当触发信号既有上升沿又有下降沿时，选择 either 触发状态复位.

复位端口是直接传过的. 如果模块的输出直接或通过一系列直接馈通的模块反馈给了这一端口，将会导致代数回路. 要解除这一回路，不通过模块的输出而是状态端口传给复位端口. 要获得模块的状态，选择显示状态端口(Show state port)核选框.

5)指定模块状态的绝对容限. 如果模型中包含有幅度变化非常大的状态时，为模型定义绝对容限不可能提供有效的误差控制. 要定义一个 Integrator 模块的状态的绝对容限，在 Absolute tolerance 参数域中输入一个数值. 如果模块有多个状态，同一数值适用于所有状态.

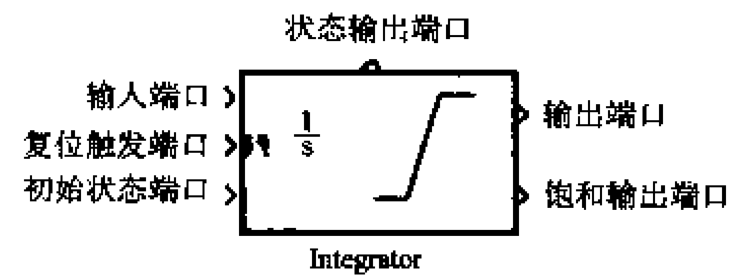


图 7.41　选择所有选项后的积分器模块图标

6)选择所有选项. 选择所有选项后，模块图标如图 7.41 所示.

(3) 模块数据类型

该模块的数据端口接受和输出双精度类型实数信号. 外部复位端口接受双精度或逻辑类型的信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.42 所示.

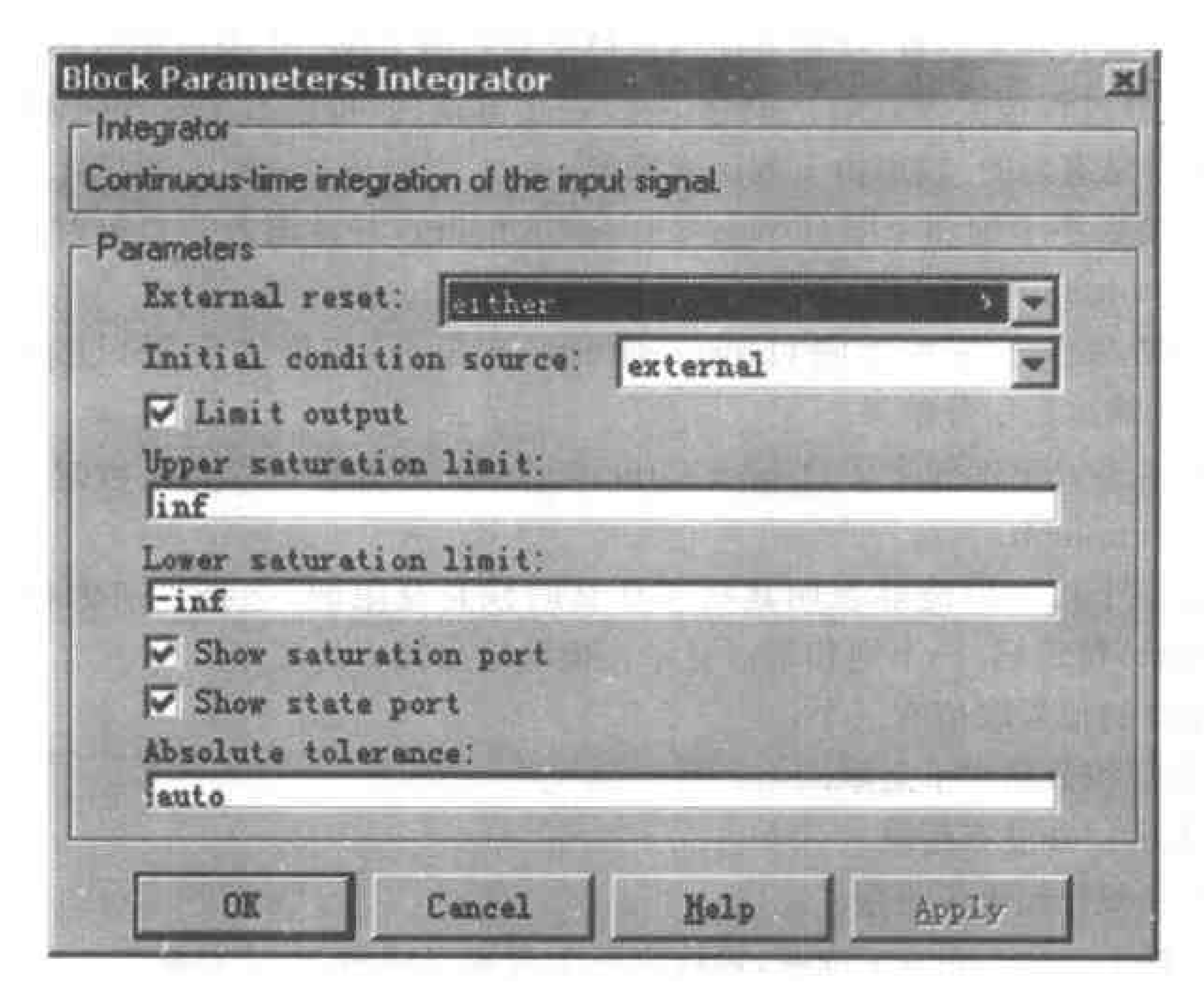


图 7.42　Integrator 模块参数对话框

1) 外部复位端口(External reset)，当触发事件发生时，复位状态为初始条件.

2) 初始条件源(Initial condition source)，获得状态初始条件源，如果设置为内部的(internal)，将从初始条件参数中获取初始条件；如果设置为外部的(external)，则从外部模块获取初始条件.

3) 初始条件(Initial condition)，指状态的初始条件. 要将初始条件源(Initial condition

source)设置为内部的(internal).

4) 限制输出(Limit output),如果选择该核选框,将限制状态值于饱和下限与饱和上限之间.

5) 饱和上限(Upper saturation limit),积分上限.缺省为 inf.

6) 饱和下限(Lower saturation limit),积分下限.缺省为−inf.

7) 显示饱和端口(Show saturation port),如果选中,在模块中加一个饱和输出端口.

8) 显示状态端口(Show state port),如果选中,在模块中加一个状态输出端口.

9) 绝对容限(Absolute tolerance),模块的绝对容限.

(5) 模块特点

1) 外部初始条件源复位端口,有直接馈通;

2) 采样时间是连续的;

3) 参数有标量扩展;

4) 状态从驱动模块继承;

5) 可向量化;

6) 如果限制输出,有过零区间.检测复位有一个;上、下饱和限各一个,离开饱和一个.

7.4.3　Memory(记忆)

(1) 模块功能

输出前一积分步的模块输入.

(2) 模块说明

Memory 模块输出它的前一积分步的输入,对它的输入信号使用一个积分步的采样和保持.

如图 7.43 所示的采样模型,演示了在一个仿真中如何显示步长.Sum 模块将由 Clock 模块产生的当前时间减去前一步的时间(由 Memory 模块生成).

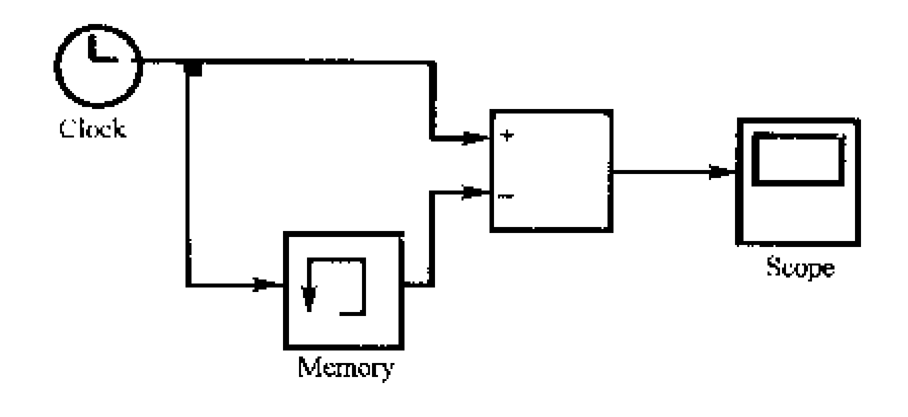


图 7.43　含有 Memory 模块的示例模型

当使用 ode15s 或者 ode113 积分时,应避免使用 Memory 模块,除非模块的输入保持不变.

(3) 模块数据类型

该模块接受任何数值类型(实数或复数)和数据类型信号,包括用户自定义类型.如果输入为用户自定义类型,则初始条件必须为 0.

(4) 模块参数对话框

该模块的参数对话框如图 7.44 所示.

Block Parameters: Memory
Memory
Apply a one integration step delay. The output is the previous input value.
Parameters
Initial condition:
0
Inherit sample time
OK Cancel Help Apply

图 7.44 Memory 模块参数对话框

1) 初始条件(Initial condition),指初始积分步的输出.

2) 继承采样时间(Inherit sample time),选中该核选框,采样时间将从驱动模块继承.

(5) 模块特点

1) 没有直接馈通;

2) 采样时间是连续的,但如果选中继承采样时间核选框,将继承采样时间;

3) 初始条件参数有标量扩展;

4) 状态为 0;

5) 可向量化;

6) 没有过零区间.

7.4.4 State-Space(状态空间)

(1) 模块功能

实现线性状态空间系统.

(2) 模块说明

State-Space 模块实现由式(7.15)定义的系统:

$$\begin{aligned} x' &= Ax + Bu \\ y &= Cx + Du \end{aligned} \tag{7.15}$$

式中 x 是状态向量,u 是输入向量,y 是输出向量. 系数矩阵 A,B,C,D 必须满足如图 7.29 所示的要求.

A 必须是 $n \times n$ 的矩阵,其中 n 是状态的个数.

B 必须是 $n \times m$ 的矩阵,其中 m 是输入的个数.

C 必须是 $r \times n$ 的矩阵,其中 r 是输出的个数.

D 必须是 $r \times m$ 的矩阵.

该模块接收一个输入并且产生一个输出. 输入向量的宽度由 B 和 D 矩阵的的列数确定. 输出向量的宽度由 C 和 D 矩阵的行数确定.

Simulink 将含有 0 的矩阵转换为稀疏矩阵,以实现更有效的乘法.

(3) 模块数据类型

该模块接受和输出双精度类型实数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.45 所示.

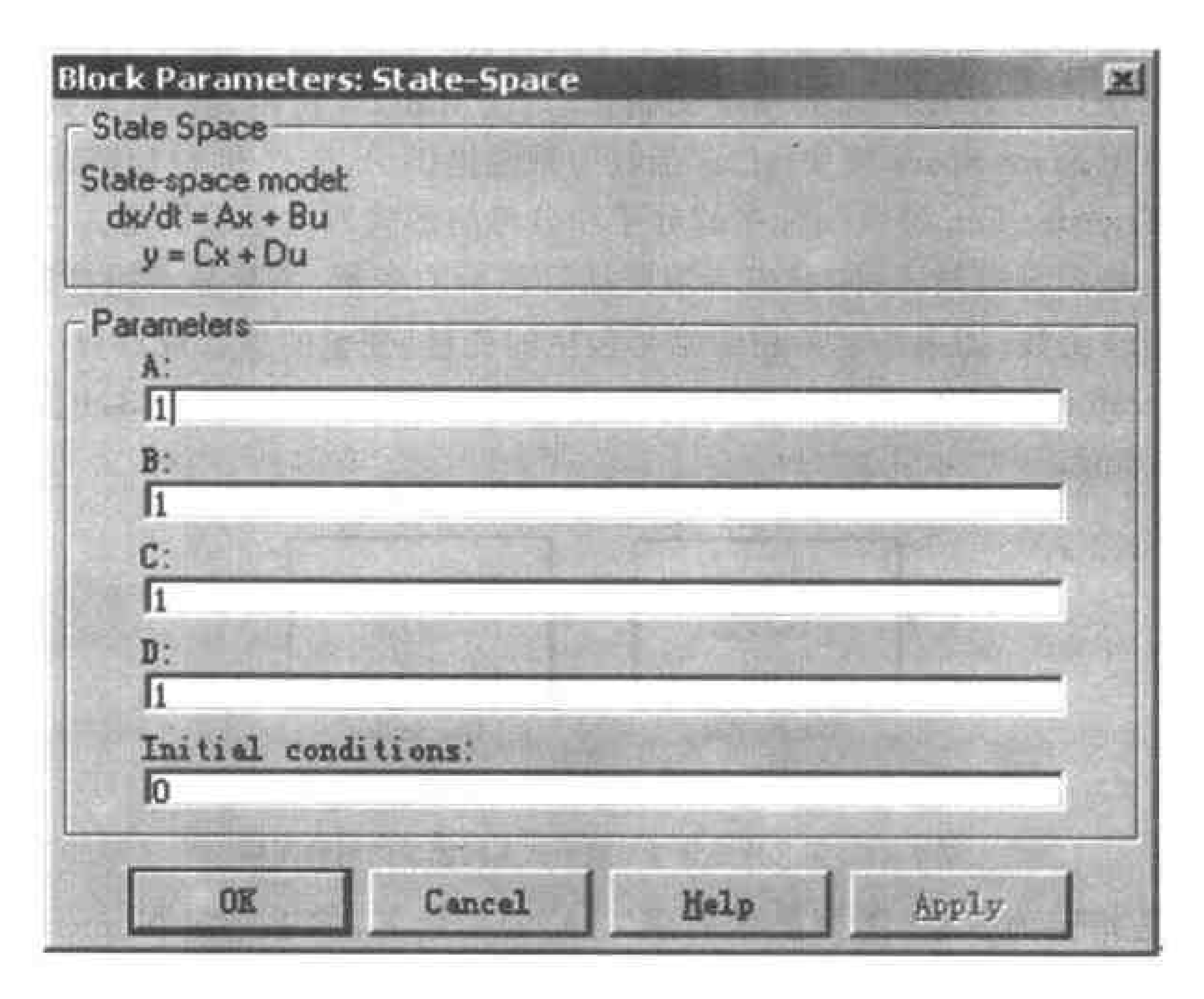


图 7.45 State-Space 模块参数对话框

1) A,B,C,D 为系数矩阵.

2) 初始条件,初始状态向量.

(5) 模块特点

1) 仅当 D 不等于 0 时,有直接馈通;

2) 采样时间是连续的;

3) 初始条件有标量扩展;

4) 状态取决于 A 的大小;

5) 可向量化;

6) 没有过零区间.

7.4.5 Transfer Fcn(传递函数)

(1) 模块功能

实现一个线性传递函数.

(2) 模块说明

Transfer Fcn 模块实现传递函数,它的输入 u 与输出 y 可以用传递函数式(7.16)表达:

$$H(s)=\frac{y(s)}{u(s)}=\frac{num(s)}{den(s)}=\frac{num(1)s^{nn-1}+num(2)s^{nn-2}+\cdots+num(nn)}{den(1)s^{nd-1}+den(2)s^{nd-2}+\cdots+den(nd)} \quad (7.16)$$

式中 nn 和 nd 分别是分子和分母的系数的个数. num 和 den 是以 s 的降幂表示的分子和分母的系数. num 可以是向量也可以是矩阵, den 必须是向量,它们都可以在对话框的相应参数域中指定. 分母的阶数必须大于或者等于分子的阶数.

模块输入是标量,输出的宽度等于分子中的行数.

初始状态预设为 0. 如果需要指定初始状态,使用 uf2ss 将传递函数转换为状态空间的形式并且使用 State-Space 模块. tf2ss 函数为系统提供 A,B,C 和 D 矩阵.

显示在 Transfer Fcn 模块图标中的分子和分母的形式取决于它们是如何指定的:

如果都是指定为表达式、向量或者用圆括号包括的变量,图标显示用指定的系数和 s 的幂组成的传递函数. 如果指定为用圆括号包括的变量,变量的值会被计算出来. 例如,如果指定 Numerator 为[3,2,1],Denominator 为(den),其中 den 是[7,5,3,1],模块的图标如图 7.46(a)所示.

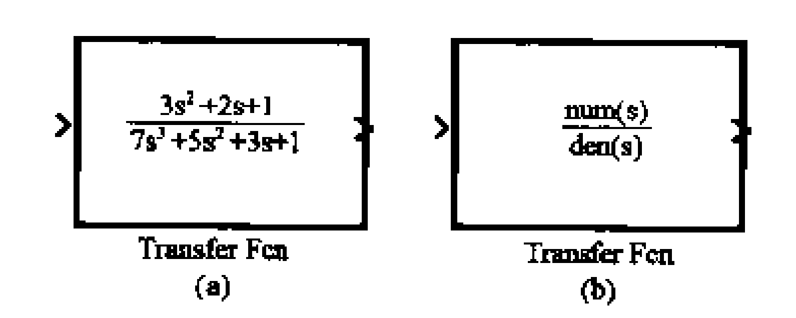


图 7.46　定义分子分母系数形式不同模块的两种图标

如果都是指定为变量,图标显示后跟"(s)"的变量名,如果指定 Numerator 为 num,Denominator 为 den,模块的图标如图 7.46(b)所示.

(3) 模块数据类型

该模块接受和输出任何数据类型的信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.47 所示.

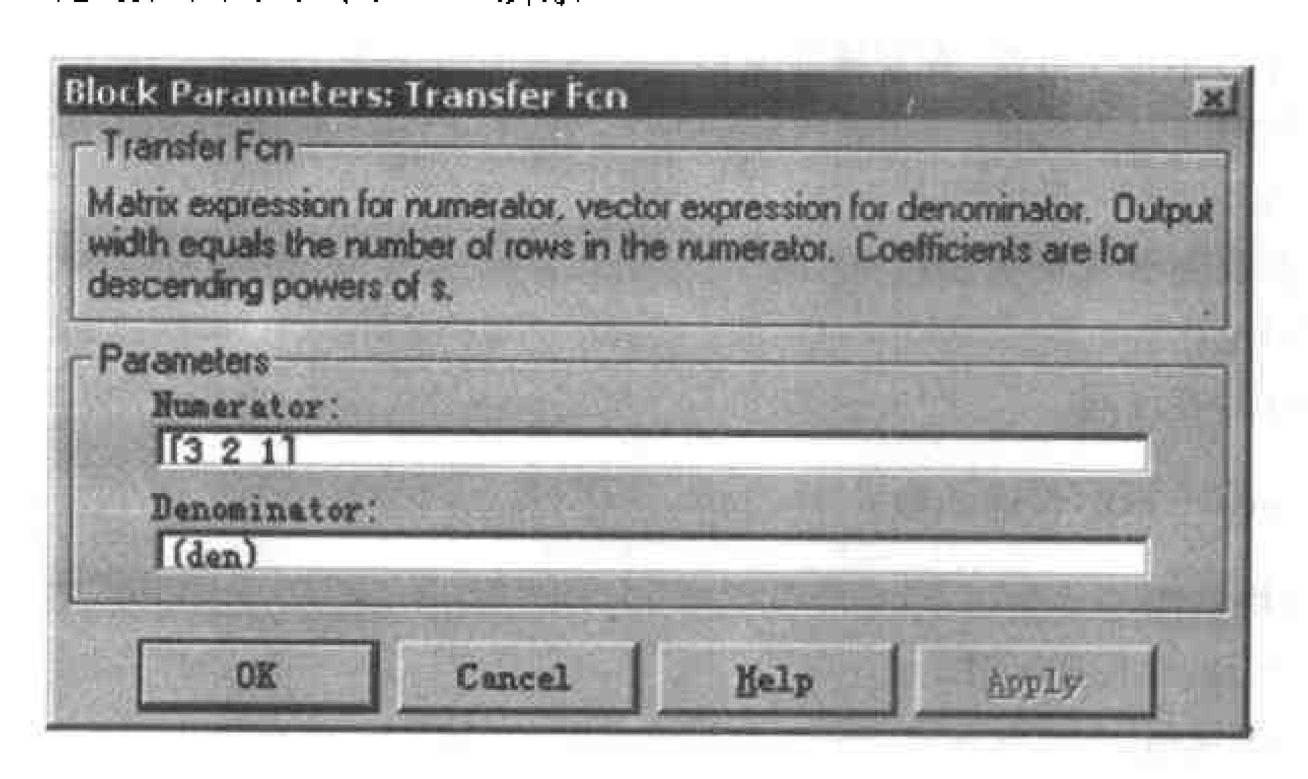


图 7.47　Transfer Fcn 模块参数对话框

1) 分子(Numerator),分子系数行向量. 一个具有多行的矩阵可以用来指定产生多个输出. 缺省值为[1].

2）分母(Denominator)，分母系数行向量. 缺省值为[1 1].

(5) 模块特点

1）仅当分子与分母参数长度相等时有直接馈通；

2）采样时间是连续的；

3）没有标量扩展；

4）状态为分母长度－1；

5）可向量化；

6）没有过零区间.

7.4.6 Transport Delay(传递延迟)

(1) 模块功能

将输入延迟一段给定的时间.

(2) 模块说明

Transport Delay 模块将输入延迟一段给定的时间. 它可用来模拟时间延迟.

在仿真开始的时候，模块输出 Initial input 参数值直到仿真时间超过 Time delay 参数值，这时模块开始产生经过延迟的输入. Time delay 参数必须是非负数.

该模块在仿真期间在缓冲区中保存输入的数据点和仿真时间，缓冲区的初始大小由 Initial buffer size 参数定义. 如果数据点的数目超过了缓冲区的大小，模块分配另外的内存并且在仿真结束后 Simulink 显示一条消息以表示总共需要的缓冲区. 因为分配内存会降低仿真速度，如果仿真速度是需要考虑的问题时则需要精心地设置这一参数值. 对于长时间的延迟，该模块会需要大量的内存，特别是对于向量化的输入.

当需要一个时刻的输出，而这一时刻与保存的输入值的时间点没有对应值，这时模块在各个点间进行线性插值. 当延迟小于步长时，模块从最后的输出点外推，这可能得到不精确的结果，因为该模块没有直接馈通，它不能使用当前的输入计算它的输出值. 要说明这一点，考虑一个步长为 1 的定步长且当前时间 t=5 的仿真. 如果延迟是 0.5，该模块需要生成 t=4.5 的输入点，因为最近保存的时间值是 t=4，因此模块进行前向外推.

Transport Delay 模块不对离散信号进行插值. 它返回 t-tdelay 时的离散值.

该模块与 Unit Delay 模块是有区别的，Unit Delay 模块只在采样点处延迟并保持输出.

使用 linmod 线性化包含有 Transport Delay 模块的模型可能会遇到麻烦.

(3) 模块数据类型

该模块接受和输出双精度类型实数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.48 所示.

1）时间延迟(Time delay)，输入信号在传给输出前被延迟的仿真时间量，该参数值不能为负. 缺省值为 1.

2）初始输入(Initial input)，指模块在仿真开始与时间延迟之间产生的输出. 缺省值为 0.

3）初始缓存大小(Initial buffer size)，初始分配内存存储的点数. 缺省值为 1024.

图 7.48　Transport Delay 模块参数对话框

(5) 模块特点

1) 没有直接馈通；

2) 采样时间是连续的；

3) 除初始缓存大小之外的所有参数和输入有标量扩展；

4) 可向量化；

5) 没有过零区间.

7.4.7　Variable Transport Delay(可变传输延迟)

(1) 模块功能

将输入延迟一段可变的时间.

(2) 模块说明

Variable Transport Delay 模块能够用来模拟可变时间的延迟. 该模块可以用来模拟有着管道的系统，管道中泵中液体的速度是可变的.

该模块有两个输入：第一个输入是传过模块的信号；第二个输入是时间延迟.

参数最大延迟(Maximum delay)定义对输入的最大延迟时间. 该模块截去超过该值的延迟时间值. 最大延迟必须大于或者等于 0. 如果延迟时间变成负数，模块将它截为 0 并发出一条警告消息.

在仿真期间，该模块保存成对的时间和输入值到内部缓冲区. 在仿真开始的时候，模块的输出是 Initial input 设定的值，直到仿真时间超过输入的延迟时间. 然后，在每一仿真步，模块输入的信号是当前仿真时间减去延迟时间时的输入信号.

当需要某一个时刻的输出，而保存的时间/输入值对中没有对应的时间，这时模块在数据点间线性地插值. 如果时间延迟小于步长，模块外推得到输出点，这时结果可能不是很准确. 该模块不能使用当前的输入值去计算它的输出值，因为它不是直接馈通的. 要说明这一点，考察一个步长为 1 的定步长仿真，当前时间是 t=5，如果延迟是 0.5，模块需要产生在 t=4.5 时的点，因为最近保存的时间值是 t=4，所以模块执行前向外推.

Variable Transport Delay 模块不对离散信号进行插值,这时,它返回的是 t-tdelay 时的离散值.

(3) 模块数据类型

该模块接受和输出双精度类型实数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.49 所示.

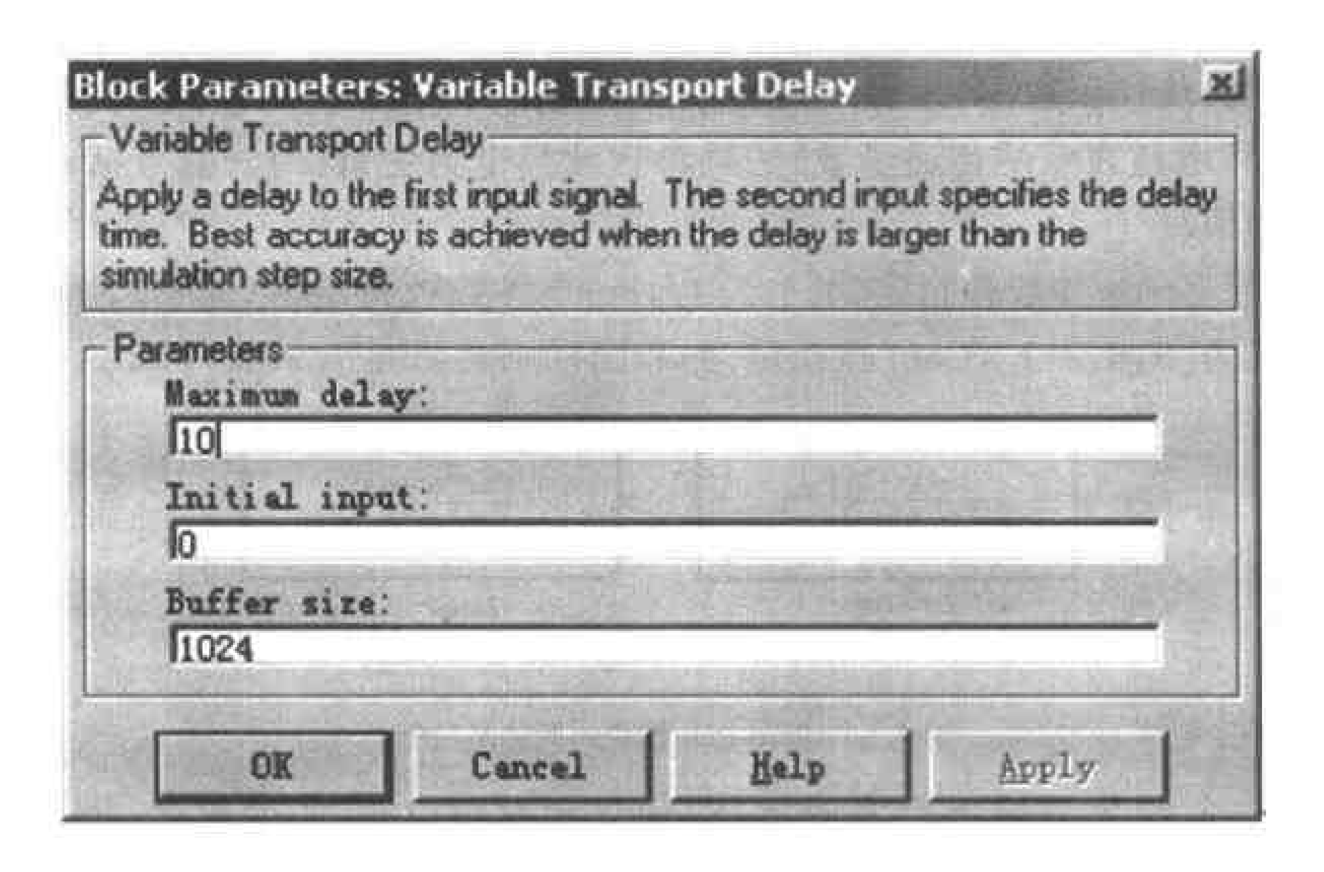


图 7.49 Variable Transport Delay 模块参数对话框

1) 最大延迟(Maximum delay),时间延迟输入的最大值.该值不能为负.缺省值为 10.

2) 初始输入(Initial input),仿真第一次超过时间延迟输入之前,模块产生的输出.缺省值为 0.

3) 缓存大小(Buffer size),模块可存储点数.缺省值为 1024.

(5) 模块特点

1) 时间延迟(第二个输入)是直接馈通;

2) 采样时间是连续的;

3) 除缓存大小之外的所有参数和输入有标量扩展;

4) 可向量化;

5) 没有过零区间.

7.4.8 Zero-Pole(零-极点)

(1) 模块功能

实现以零-极点的形式指定的传递函数.

(2) 模块说明

Zero-Pole 模块实现以拉普拉斯操作符 s 的形式指定零点、极点和增益的系统.

传递函数可以用零点-极点-增益的形式表示,对于 MATLAB 中的单输入单输出系

统，表达式如下：

$$H(s)=K\frac{Z(s)}{P(x)}=K\frac{(s-Z(1))(s-Z(2))\cdots(s-Z(m))}{(s-P(1))(s-P(2))\cdots(s-P(n))} \tag{7.17}$$

式中 Z 表示零点向量，P 表示极点向量，K 是增益. Z 可以是向量也可以是矩阵，P 必须是向量，K 可以是标量或者是长度等于 Z 的行数的向量. 极点的个数必须大于或者等于零点的个数. 如果零极点是复数，它们必须是共轭复数对.

模块的输入输出宽度等于零点矩阵的行数. Zero-Pole 模块在它的图标中根据指定的参数来显示它的传递函数. 如果每一个参数都被指定为表达式或者向量，图标显示用指定的零点、极点和增益表示的传递函数. 如果参数被指定为变量（用圆括号括起来），图标中将显示变量的值.

例如，如果 Zeros 参数被指定为[4,2,1]，Poles 参数被指定为(poles)，而 poles 在工作空间中被定义为[9,6,3,1]，Gain 被指定为 G，图标则会如图 7.50(a)所示.

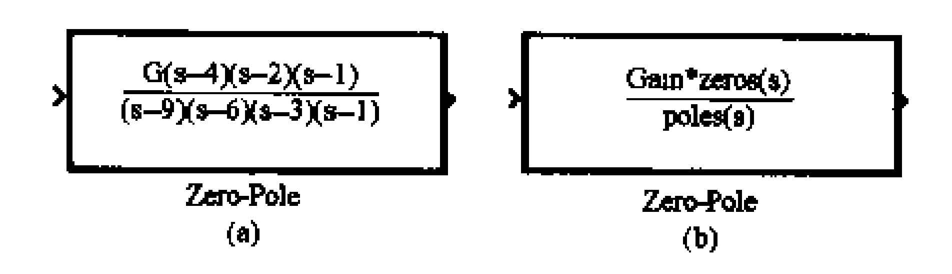


图 7.50　Zero-Pole 模块的两种图标形式

如果每一个参数都被指定为变量，图标显示后跟"(s)"的变量名. 例如，如果指定 Zeros 为zeros，指定 Poles 为 poles，指定 Gain 为 gain，则图标看起来会如图 7.50(b)所示.

(3) 模块数据类型

该模块接受双精度类型实数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.51 所示.

Block Parameters: Zero-Pole1

Zero-Pole

Matrix expression for zeros. Vector expression for poles and gain. Output width equals the number of columns in zeros matrix, or one if zeros is a vector.

Parameters

Zeros:
[1]

Poles:
[0 -1]

Gain:
[1]

OK　Cancel　Help　Apply

图 7.51　Zero-Pole 模块参数对话框

1) 零点(Zeros),指零矩阵. 缺省值为[1].
2) 极点(Poles),指极点向量. 缺省为[0 −1].
3) 增益(Gain),指增益向量. 缺省值为[1].
(5) 模块特点
1) 仅当零点与极点参数长度相等时有直接馈通;
2) 采样时间是连续的;
3) 没有标量扩展;
4) 状态为极点长度;
5) 不可向量化;
6) 没有过零区间.

7.5　Math 库中的模块

Math(数学)库中包含的模块如表 7.6 所列,各个模块的图标如图 6.6 所示.

表 7.6　Math 库中的各模块

模块名	功　能
Abs	输出输入的绝对值
Algebraic Constraint	将输入信号抑制为 0
Combinatorial Logic	实现真值表
Complex to Magnitude-Angle	输出复数输入信号的相角和幅值
Complex to Real-Imag	输出复数输入信号的实部和虚部
Derivative	输出输入的时间导数
Dot Product	产生点积
Gain	将模块的输入乘以一个数值
Logical Operator	对输入执行指定的逻辑操作
Magnitude Angle to Complex	由相角和幅值输入输出一个复数信号
Math Function	执行一个数学函数
Matrix Gain	将输入乘以一个矩阵
MinMax	输出输入的最小或最大值
Product	产生模块各输入的简积或商
Real-Imag to Complex	由实部和虚部输入输出一实数信号
Relational Operator	对输入执行指定的关系操作
Rounding Function	执行圆整函数
Sign	指明输入的符号
Slider Gain	使用滑动器改变标量增益
Sum	生成输入的和
Trigonometric Function	执行三角函数

7.5.1　Abs(绝对值)

(1) 模块功能

输出输入的绝对值.

(2) 模块说明

Abs 模块产生的输出是输入的绝对值.该模块接受一个输入并且产生一个输出.

(3) 模块数据类型

该模块接受双精度类型实数或复数值输入,产生一个双精度类型的实数输出.

(4) 模块参数对话框

该模块的参数对话框如图 7.52 所示.

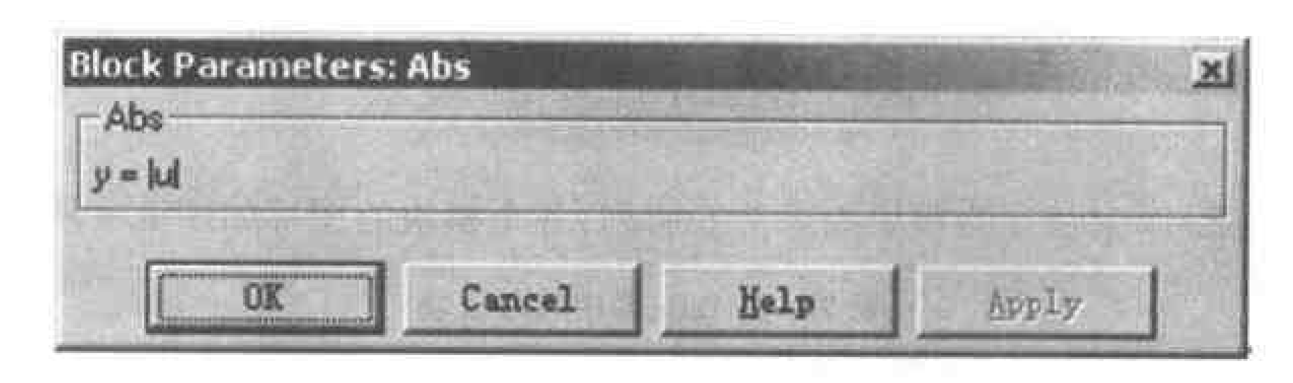


图 7.52　Abs 模块参数对话框

(5) 模块特点

1) 直接馈通;

2) 采样时间由驱动模块继承;

3) 标量扩展 N/A;

4) 可向量化;

5) 有过零区间.

7.5.2　Algebraic Constraint

(1) 模块功能

抑制输入信号为 0.

(2) 模块说明

Algebraic Constraint 模块将输入信号 f(z)抑制为 0 并且输出一个代数的状态 z.该模块输出使输入为 0 时所必需的值.输出必须通过一些反馈影响输入.这样就可以为一阶微分/代数系统(DAEs)指定代数方程.

缺省情况下,参数 Initial guess(初始估计)的值为 0.可以通过给代数状态 z 提供一个接近于计算结果的 Initial guess 来提高代数回路计算器的效率.

例 7.1　如图 7.53 所示的模型可以用来计算如下的方程:

$$z2+z1=2$$

$$z2-z1=2$$

上式的计算结果是 $z2=1$,$z1=0$,正如 Display 模块显示的那样.

(3) 模块数据类型

该模块接受和输出双精度类型实数值.

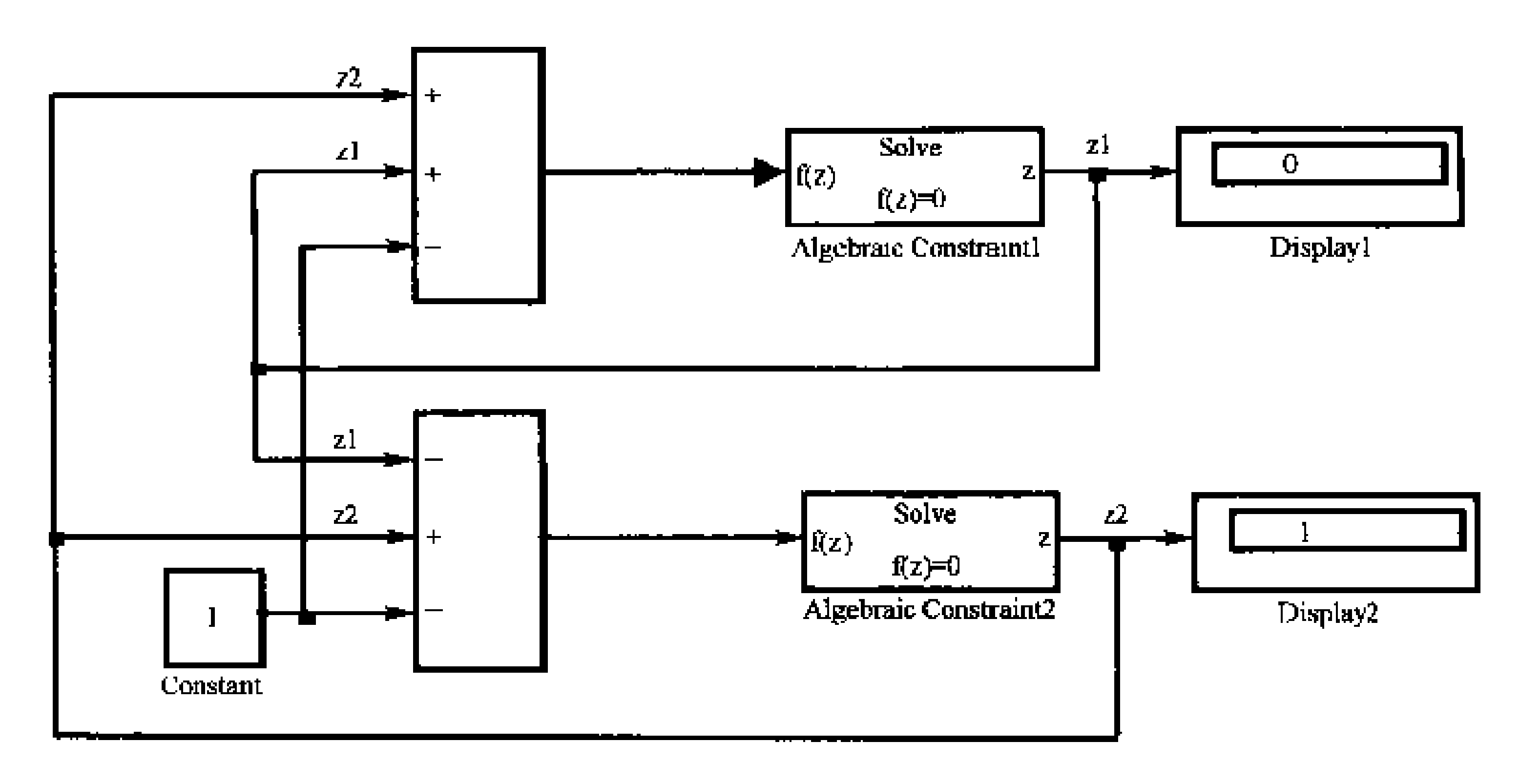


图 7.53 包含 Algebraic Constraint 模块的示例模型

(4) 模块参数对话框

该模块的参数对话框如图 7.54 所示.

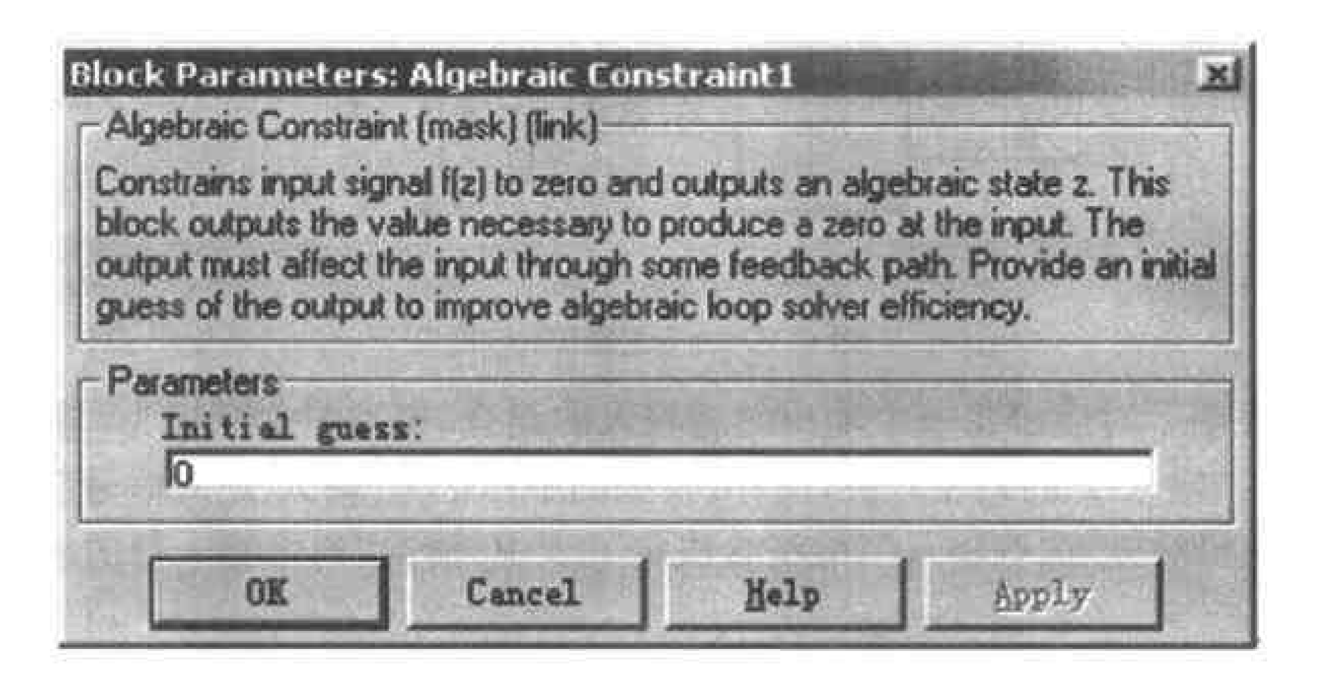


图 7.54 Algebraic Constraint 模块参数对话框

初始估计(Initial guess)指对结果的初始估计值.缺省值为 0.

(5) 模块特点

1) 直接馈通;

2) 采样时间由驱动模块继承;

3) 没有标量扩展;

4) 可向量化;

5) 没有有过零区间.

7.5.3 Combinatorial Logic(组合逻辑)

(1) 模块功能

实现真值表.

(2) 模块说明

Combinatorial Logic 模块可用于在对可编程逻辑阵列(PLAs)、逻辑电路、决策表和其它的逻辑表达式进行建模时,实现一个标准的真值表.

可以指定一个定义模块所有可能的输出矩阵作为 Truth table(真值表)参数. 矩阵的每一行包含有对应输入元素的不同组合的输出. 必须为输入的每一种可能的组合指定一个输出. 每一个输出都可以是任何数字量;它不局限于取 0 和 1 这两个值. 矩阵的列数就是模块的输出数.

模块输入的个数与矩阵的行数之间的关系是:行数 $=2^{输入数}$.

通过计算一个以输入向量各元素所在的某一行的索引为参数的表达式,Simulink 得到要返回的输出矩阵中对应于该行元素的值. Simulink 将输入向量为 0 的各元素用 0 代替,不为 0 的各元素用 1 代替,然后用下面的表达式计算得到行索引,对于一个有 m 个元素的输入向量 u:

$$行索引 = 1 + u(m)\times 2^0 + u(m-1)\times 2^1 + \cdots + u(1)\times 2^{m-1} \quad (7.18)$$

例 7.2 建立一个有两个输入的与函数的模型,当两个输入的元素都为 1 时返回 1,否则返回 0. 要实现这一函数,指定参数 Truth table 的值为[0;0;0;1]. 模型中给组合逻辑提供输入和输出的部分如图 7.55 所示.

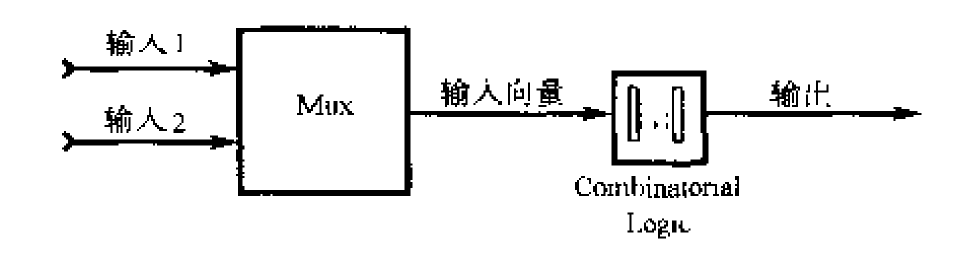


图 7.55 组合逻辑模块示例

表 7.7 表明了对应于每一个输出的各输入的组合. 模型图中标记为"输入 1"的信号与表中标记为"输入 1"的列相对应,同样,模型图中标记为"输入 2"的信号与表中标记为"输入 2"的列相对应. 这些输入值的组合决定了表中"输出"列中哪一行的值作为模块的输出.

表 7.7 两输入与门真值表

行	输入 1	输入 2	输出
1	0	0	0
2	0	1	0
3	1	0	0
4	1	1	1

如果输入向量是[1 0],通过用式(7.18)计算得到模块的输出,应该是"输出"的第三$(1+0\times 2^0+1\times 2^1)$行,即输出值应为 0.

例 7.3 有三个输入的电路例子:两个相加的输入位 a 和 b,一个进位输入 c. 它有两个输出,进位输出 c′与相加和位 s. 表 7.8 是这一电路的真值表和与输入值的每一种组合相对应的输出.

表 7.8　电路的真值表

输　入			输　出	
a	b	c	c′	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

要用 Combinatorial Logic 模块实现这一加法器，在对话框的 Truth table 参数域中输入由 c′和 s 列组成的 8×2 的矩阵.

时序电路(也就是有状态的电路)同样可以通过给模块的状态引入另外一个输入，并且将模块的输出反馈给该输入来实现.

(3) 模块数据类型

该模块接受逻辑或双精度类型实数信号，输出与输入类型一致. 真值表元素也可以是逻辑或双精度类型. 如果元素为双精度类型，可以是任何值，不仅仅是“逻辑”(0 或 1)值. 如果真值表元素数据类型不同于输出信号的类型，Simulink 在计算输出前转换真值表为输出类型.

(4) 模块参数对话框

该模块的参数对话框如图 7.56 所示.

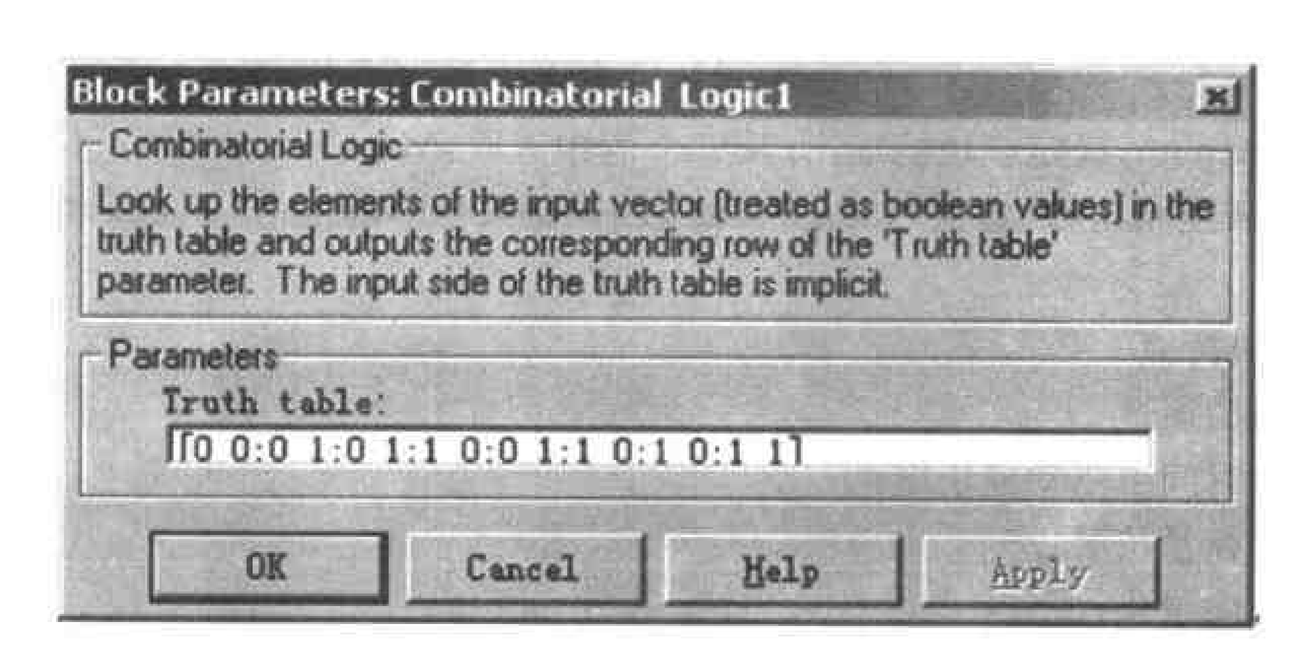


图 7.56　Combinatorial Logic 模块参数对话框

真值表(Truth table)指输出矩阵. 每列与输出向量的一个元素相对应，每行与直值表的一行相对应.

(5) 模块特点

1) 直接馈通；

2) 采样时间由驱动模块继承；

3) 没有标量扩展;

4) 可向量化,输出宽度是真值表参数的列数;

5) 没有过零区间.

7.5.4 Complex to Magnitude-Angle

(1) 模块功能

计算复数信号的幅值和相角.

(2) 模块说明

该模块接受一复数信号,输出输入信号的幅值和相角. 输出实数值. 输入可以是复数信号向量,此时输出也是向量. 幅值向量包含对应复数输入元素的幅值,角度输出包含相应复数输入元素的相角.

(3) 模块数据类型

该模块接受双精度类型复数值信号,输出双精度类型的实数值.

(4) 模块参数对话框

该模块的参数对话框如图 7.57 所示.

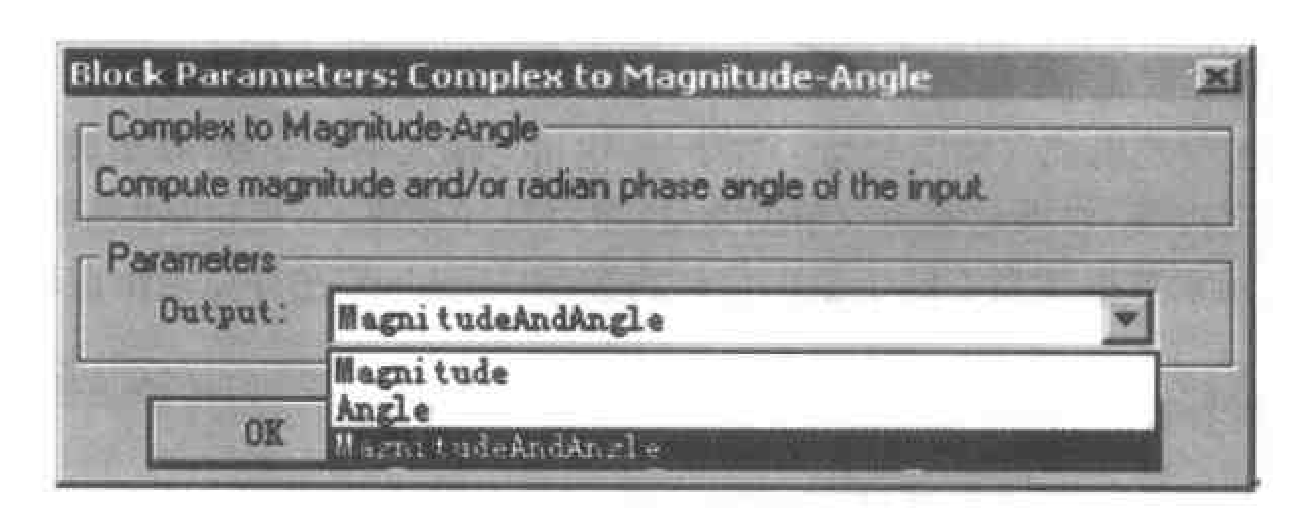


图 7.57 Complex to Magnitude-Angle 模块参数对话框

输出(Output),确定模块的输出. 从三个值中选取:MagnitudeAndAngle (输出输入信号的幅值和弧度相角,缺省值),Magnitude (输出输入信号的幅值),Angle (输出输入的弧度相角).

(5) 模块特点

1) 直接馈通;

2) 采样时间由驱动模块继承;

3) 没有标量扩展;

4) 不可向量化;

5) 没有过零区间.

7.5.5 Complex to Real-Imag

(1) 模块功能

输出一个复数输入信号的实部和虚部.

(2) 模块说明

该模块接受双精度类型的复数值信号，输出输入信号的实部、虚部. 输入可以是复数信号向量，此时输出也是向量.

(3) 模块数据类型

该模块接受双精度类型复数值信号输入，输出双精度类型的实数值.

(4) 模块参数对话框

该模块的参数对话框如图 7.58 所示.

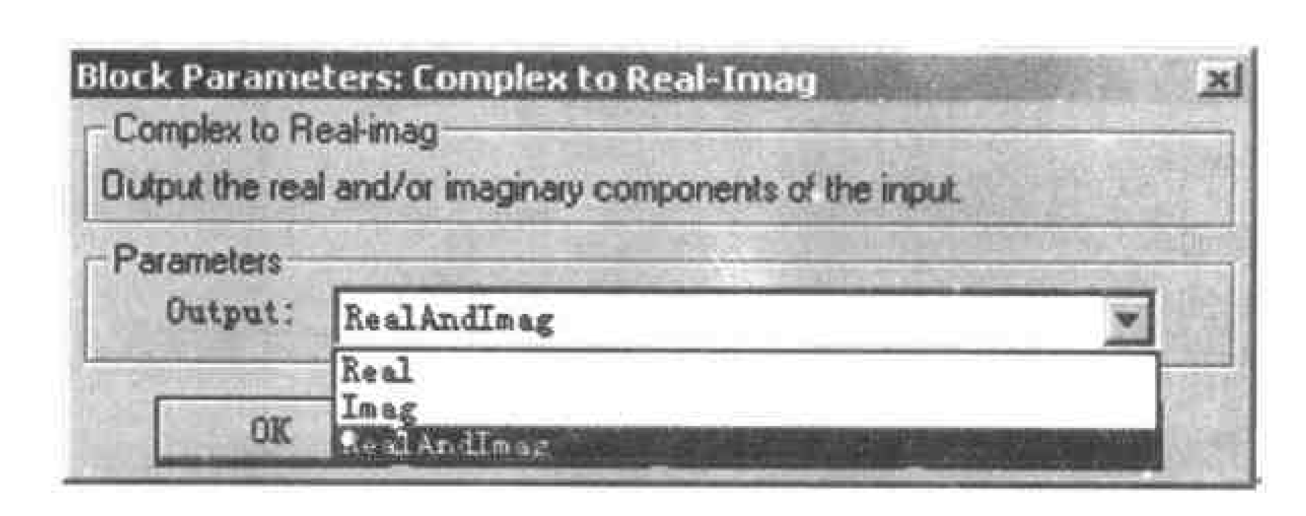


图 7.58　Complex to Real-Imag 模块参数对话框

输出(Output)，确定模块的输出. 可选择为：RealAndImag(输出输入信号的实部和虚部，缺省值)，Real(输出输入信号的实部)，Imag(输出输入信号的虚部).

(5) 模块特点

1) 直接馈通；

2) 采样时间由驱动模块继承；

3) 没有标量扩展；

4) 不可向量化；

5) 没有过零区间.

7.5.6 Dot Product(点乘)

(1) 模块功能

产生点乘积.

(2) 模块说明

Dot Product 模块产生它的两个输入向量的点乘积. 标量输出 y 等于 MATLAB 的运算：

$$y = u1' * u2$$

其中 $u1$ 和 $u2$ 表示向量输入. 如果两个输入都是向量，它们的长度必须相同. 输入向量的元素可以是实数或复数，输出的类型依赖于输入的类型.

要执行元素对元素的乘积而不将它们相加，使用 Product 模块.

(3) 模块数据类型

该模块接受和输出双精度类型信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.59 所示.

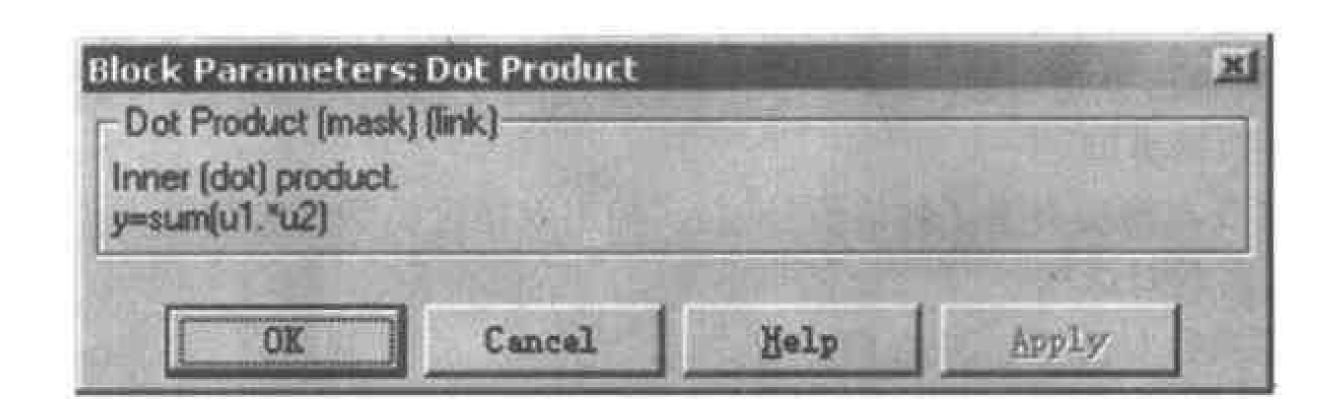


图 7.59　Dot Product 模块参数对话框

(5) 模块特点

1) 直接馈通;

2) 采样时间由驱动模块继承;

3) 可以标量扩展;

4) 状态 0;

5) 可向量化;

6) 没有过零区间.

7.5.7　Gain(增益)

(1) 模块功能

将模块输入乘以一个数.

(2) 模块说明

Gain 模块将其输入乘以一个指定的常数、变量或表达式作为它的输出.

增益既可以是数值、变量或表达式.要将输入乘以矩阵,可以用 Matrix Gain 模块.

如果模块的图标足够大,Gain 模块在图标中显示在 Gain 参数域中输入的参数.如果增益是作为变量输入的,模块显示变量的名称,如果变量指定在括号内,每一次重画模块时都要重新计算变量的值并显示变量的值.如果 Gain 参数的值太长而不能显示在模块中时,会显示字符串-K-.

要想在仿真时能够使用一个滑块控件改变增益时,可以使用 Slider Gain 模块.

(3) 模块数据类型

该模块接受除逻辑类型外的任何类型实数或复数值标量、向量,输出与输入类型相同.输入向量元素必须类型相同.增益参数可以是任何类型的实数或复数值标量、向量.

(4) 模块参数对话框

该模块的参数对话框如图 7.60 所示.

1) 增益(Gain),指定为一个标量、向量、变量名或表达式.缺省值为 1.没有指定的情况下,增益参数为双精度.

2) 整数溢出饱和(Saturate on integer overflow),如果选择该核选框,在整数溢出时输出模块的饱和值.如果输出数据类型是整数类型,模块输出其输出类型可表达的最大值,其绝对值意义上小于最大表达值.如果没有指定该选项,将采用仿真参数对话框中的论断页面中的数据溢出(Data overflow)事件.

(5) 模块特点

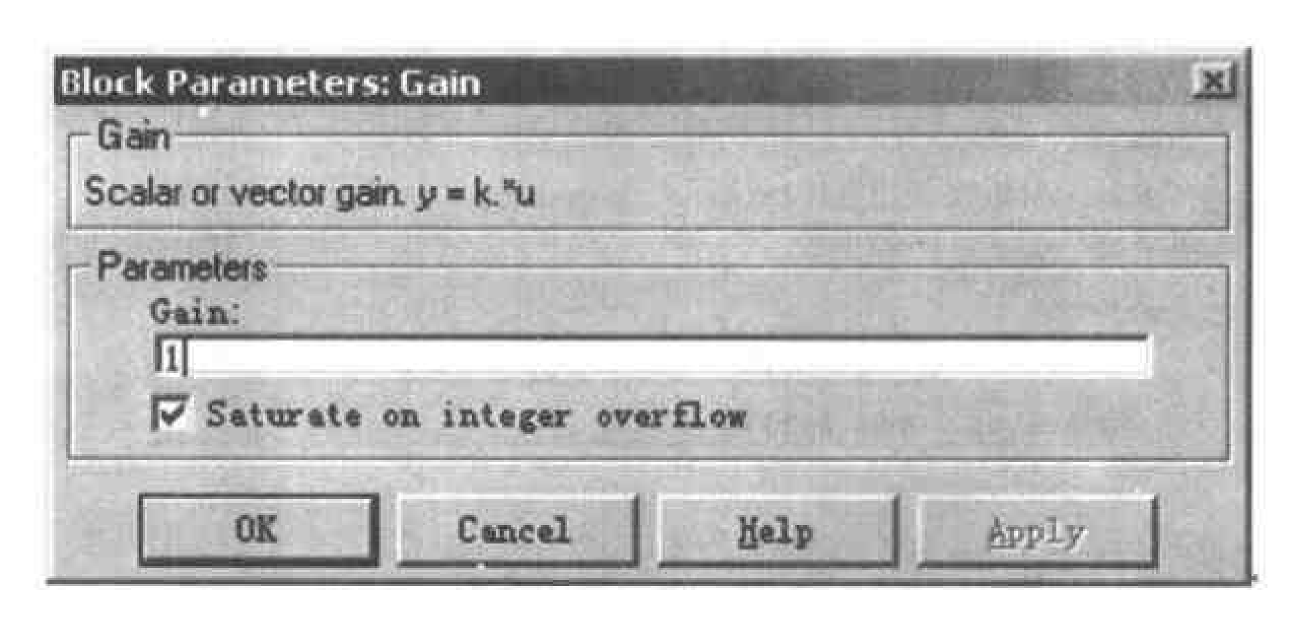


图 7.60　Gain 模块参数对话框

1）直接馈通；

2）采样时间由驱动模块继承；

3）输入和增益参数可以标量扩展；

4）状态 0；

5）可向量化；

6）没有过零区间.

7.5.8　Logical Operator（逻辑运算）

（1）模块功能

对输入执行指定的逻辑运算.

（2）模块说明

Logical Operator 模块对其输入执行这样一些逻辑运算：AND，OR，NAND，NOR，XOR 和 NOT. 输出取决于输入的数目，它们的向量大小和选用的操作符如果为 TRUE，则输出为 1，如果为 FALSE，则输出为 0. 模块的图标显示了所用的操作符.

对于两个或多个输入，模块对所有的输入执行逻辑操作. 如果输入是向量，将对各个向量的对应元素执行逻辑操作并生成一个输出向量.

对于一个单一的向量输入，模块对向量的所有元素执行逻辑操作（除 NOT 外）. NOT 操作符只接受一个输入，该输入可以是标量也可以是向量，如果输入是向量，输出向量中元素的个数与输入向量的元素的个数相同，输出向量的元素是输入向量对应元素的非.

（3）模块数据类型

该模块接受逻辑类型信号. 除非逻辑兼容模式是激活的，此时可以接受双精度类型输入，非零的输入被看作是 TRUE(1)，值为零的输入被看作是 FALSE(0). 所有输入必须类型相同，输出与输入类型一致.

（4）模块参数对话框

该模块的参数对话框如图 7.61 所示.

1）运算符(Operator)，模块要使用的逻辑运算符，从该参数的选项菜单中选取.

2）输入端口数(Number of input ports)，模块的输入数. 其数量必须与所选的运算符相符.

（5）模块特点

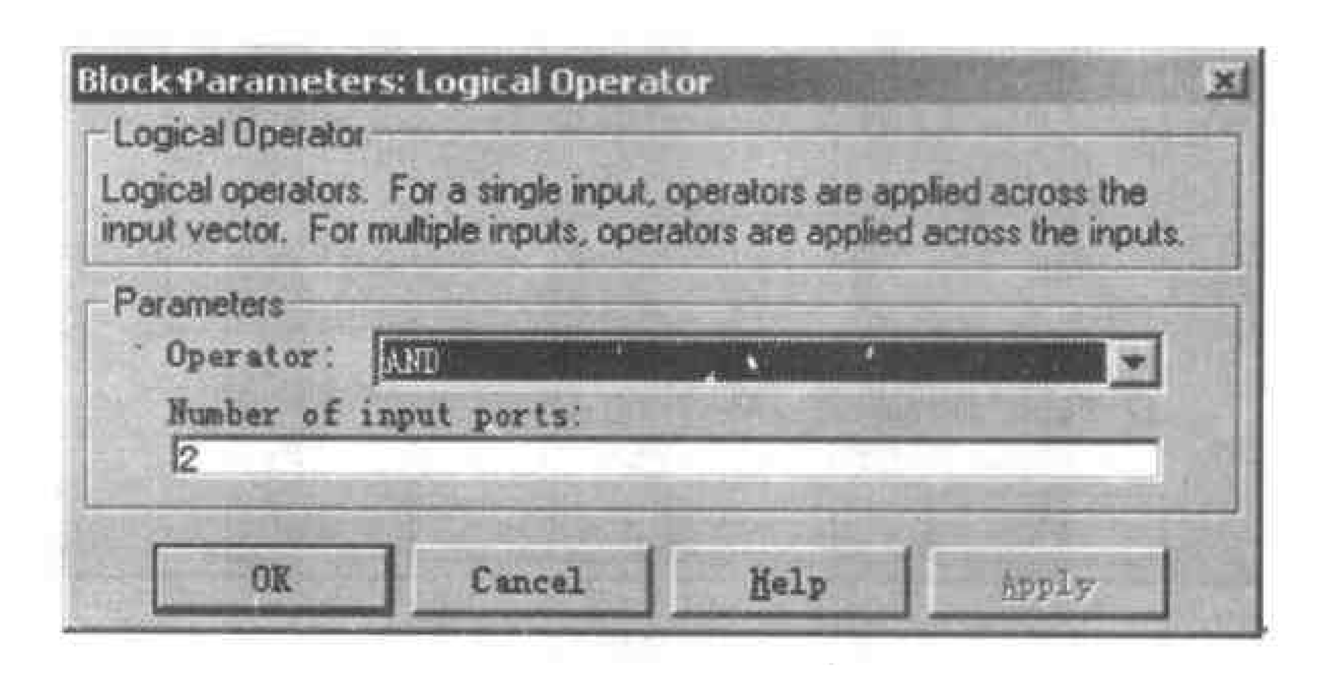


图 7.61　Logical Operator 模块参数对话框

1）直接馈通；

2）采样时间由驱动模块继承；

3）输入可以标量扩展；

4）可向量化；

5）没有过零区间.

7.5.9　Magnitude-Angle to Complex

（1）模块功能

转换幅值和(或)相角信号为复数信号.

（2）模块说明

该模块将幅值和(或)相角输入转换为复数值信号.输入必须是双精度类型实数值信号.相角以弧度为单位.输入可以是大小相同的向量,或者一个输入是一个向量,另一个是标量.幅值输入向量和相角输入向量分别映射复数输出的相应元素.如果其中一个输入为标量,它将映射输出的所有元素.

（3）模块数据类型

该模块输入为双精度类型实数值信号,输出信号的数据类型为双精度复数值.

（4）模块参数对话框

该模块的参数对话框如图 7.62 所示.

1）输入(Input),指定输入种类:幅值(Magnitude)输入,相角(Angle)输入,幅值和相角(MagnitudeAndAngle)输入(缺省值).

2）相角(Angle),如果选择幅值(Magnitude)输入,给输出指定一常数弧度相角.缺省值为 0.

3）幅值(Magnitude),如果选择相角(Angle)输入,给输出指定一常数幅值.缺省值为 0.

（5）模块特点

1）直接馈通；

2）采样时间由驱动模块继承；

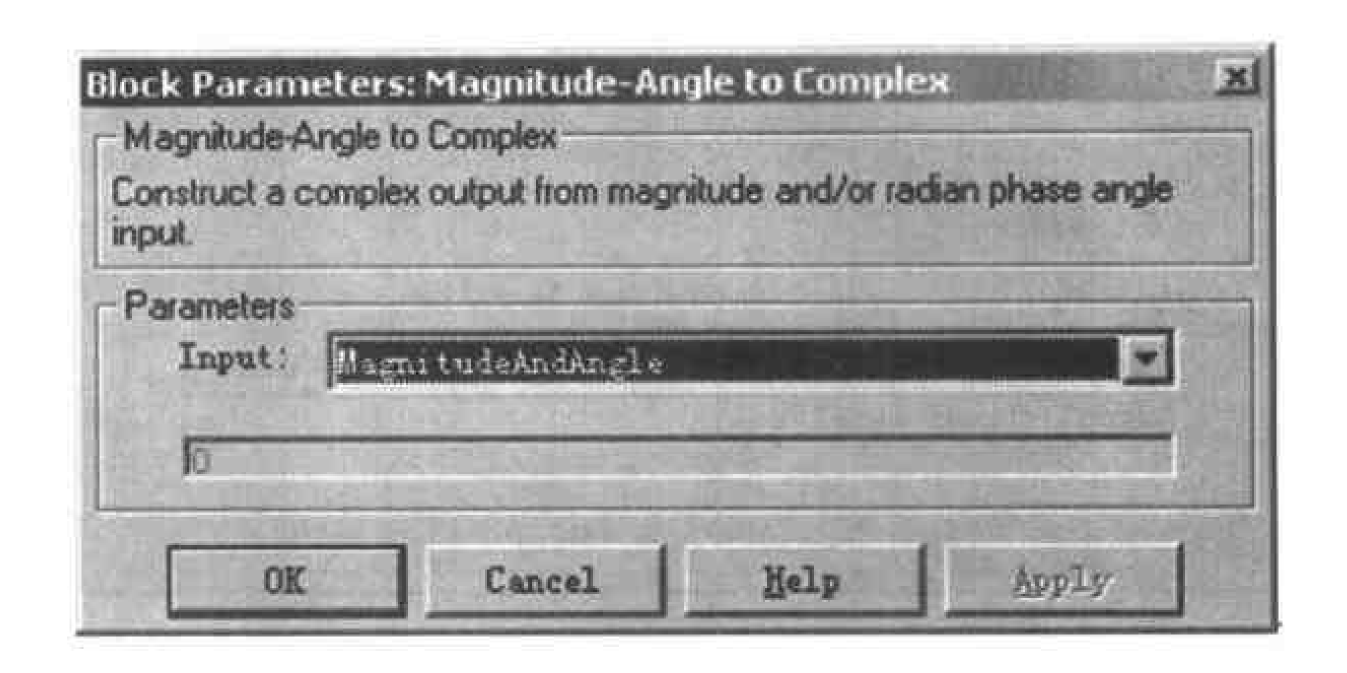


图 7.62　Magnitude-Angle to Complex 模块参数对话框

3）输入可以标量扩展；

4）可向量化；

5）没有过零区间.

7.5.10　Math Function（数学函数）

（1）模块功能

执行一个数学函数.

（2）模块说明

Math Function 模块执行许多普通的数学函数.

可以从 Function 列表框中的这些函数中选择一个：exp，log，10ˆu，log10，magnitudeˆ2，square，sqrt，pow，reciprocal，hypot，rem 和 mod. 该模块的输出是对输入执行指定函数运算的结果.

函数的名字显示在模块的图标中. Simulink 自动地画出适当数目的输入端口.

需要输出向量化的输出时应使用 Math Fucntion 模块而不是 Fcn 模块，因为 Fcn 模块只能产生标量输出.

（3）模块数据类型

该模块接受实数或复数值信号或双精度类型信号向量. 输出信号类型依据输出信号类型参数设定，为实数或复数.

（4）模块参数对话框

该模块的参数对话框如图 7.63 所示.

1）函数（Function），选择采用的数学函数.

2）输出信号类型（Output signal type），选择数学函数模块输出信号的类型为：实数（real）、复数（complex）或自动（auto）. 表 7.9 列出了各函数的输入类型与指定输出信号类型的对应关系.

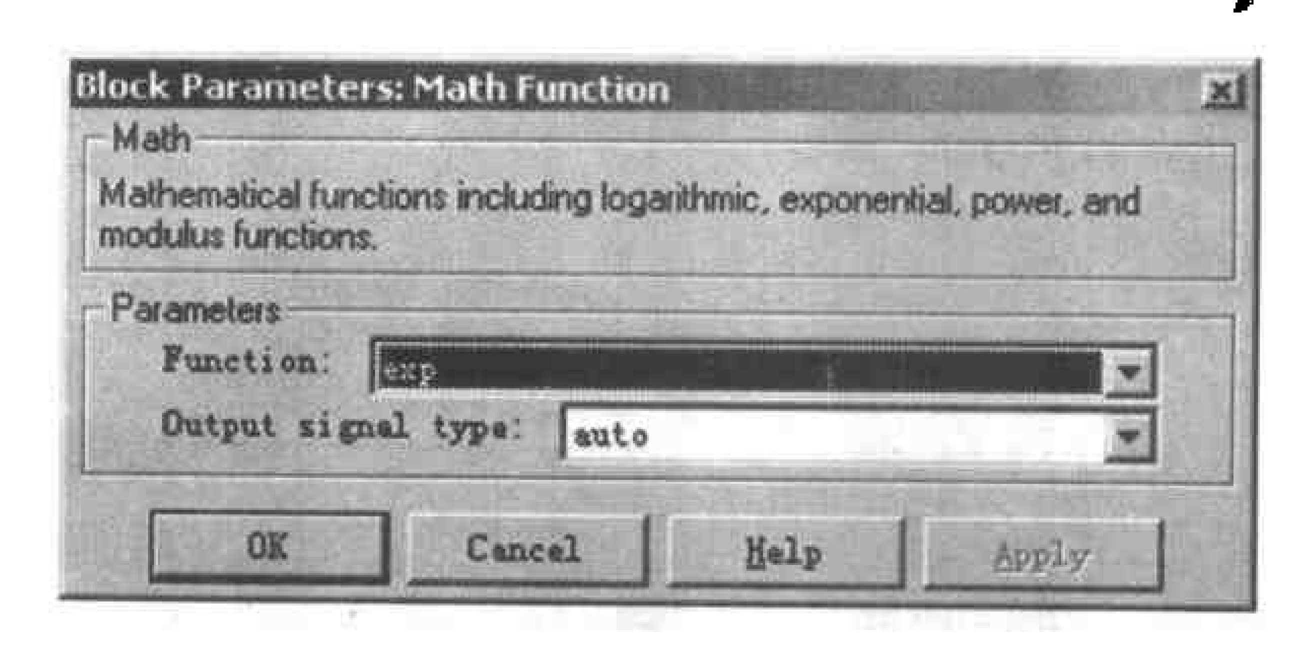


图 7.63　Math Fucntion 模块参数对话框

表 7.9　函数及输入类型与输出信号类型的关系

函　　数	输入	输出信号类型		
	信号	Auto(自动)	Real(实数)	Complex(复数)
exp, log, 10^u, squre, sqrt, pow, reciprocal, conjugate	实数	实数	实数	复数
	复数	复数	错误	复数
Magnitude squared	实数	实数	实数	复数
	复数	实数	实数	复数
hypot, rem, mod	实数	实数	实数	复数
	复数	错误	错误	错误

(5) 模块特点

1) 直接馈通；

2) 采样时间由驱动模块继承；

3) 可以标量扩展；

4) 可向量化；

5) 没有过零区间.

7.5.11　Matrix Gain(矩阵增益)

(1) 模块功能

将输入乘以一个矩阵.

(2) 模块说明

Matrix Gain 模块实现一个矩阵增益. 它的输出是向量输入与一个指定矩阵相乘的结果：

$$y = Ku \tag{7.19}$$

式中 K 是增益，u 是输入.

如果指定的矩阵有 m 行 n 列，那么模块的输入应该是一个长度为 n 的向量，输出是长度为 m 的向量.

模块的图标中通常显示 K.

如果矩阵包含有零，Simulink 将矩阵增益转换为稀疏矩阵以实现更为有效的乘法.

(3) 模块数据类型

该模块接受和输出双精度类型实数值信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.64 所示.

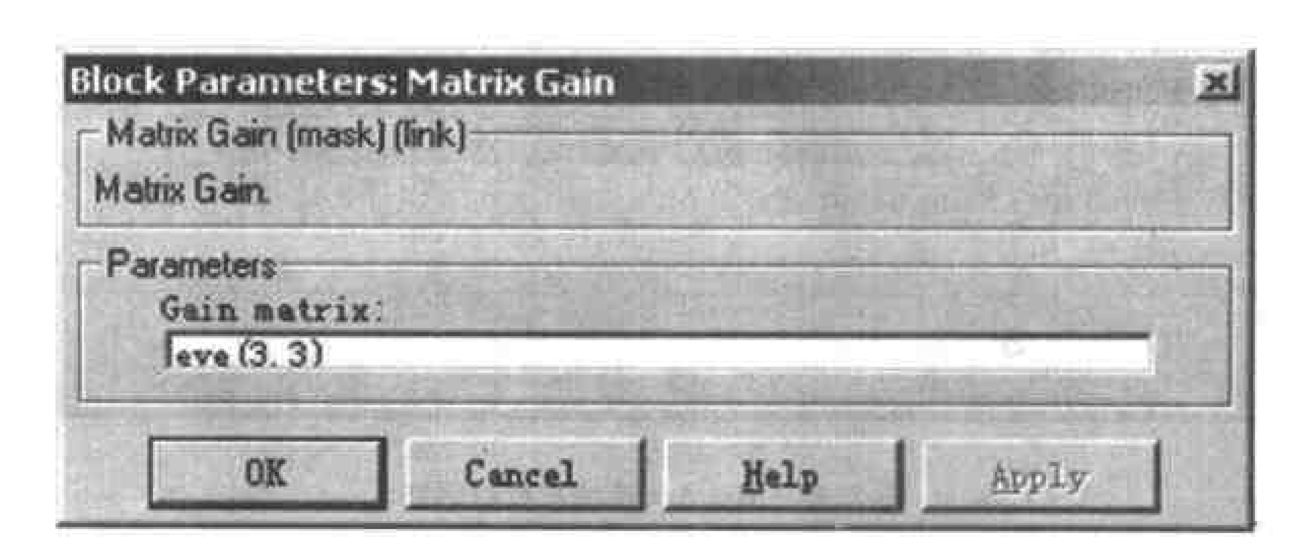


图 7.64 Matrix Gain 模块参数对话框

增益矩阵(Gain matrix),增益指定为矩阵.缺省值为 eye(3,3).

(5) 模块特点

1) 直接馈通;

2) 采样时间连续;

3) 不可标量扩展;

4) 状态 0;

5) 可向量化;

6) 没有过零区间.

7.5.12 MinMax(最小最大值)

(1) 模块功能

输出输入值的最小值或最大值.

(2) 模块说明

MinMax 模块输出其输入的最小元素或者最大元素.可以从 Function 参数列表中选择一个选项,以使用相应的函数:min 或 max.

如果模块有一个输入端口,模块的输出是一个标量,它是输入向量的最小或者最大元素.

如果模块有多个输入端口,模块对各个输入向量进行元素对元素的比较,模块输出向量的每一个元素是各个输入向量对应元素相比较的结果.

(3) 模块数据类型

该模块接受和输出双精度类型实数值信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.65 所示.

1) 函数(Function),应用于输入的函数(min 或 max).

2) 输入端口数(Number of input ports),模块的输入数.

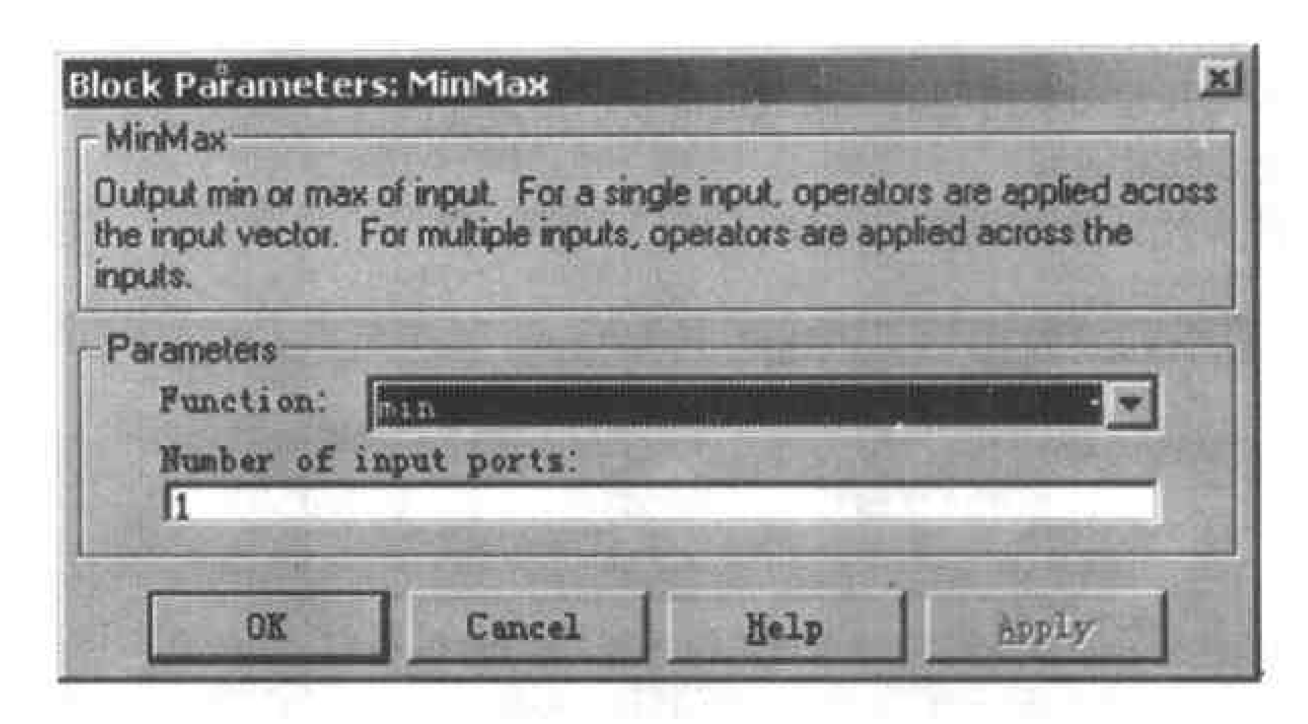


图 7.65　MinMax 模块参数对话框

(5) 模块特点

1) 直接馈通；

2) 采样时间由驱动模块继承；

3) 输入可以标量扩展；

4) 状态 0；

5) 可向量化；

6) 有过零区间，检测最小或最大值.

7.5.13　Product(乘积)

(1) 模块功能

产生模块的输入的乘积或者商.

(2) 模块说明

Product 模块对其输入进行乘或者除. 是乘还是除取决于输入数(Number of inputs)参数的值.

如果 Number of inputs 参数的值是×和÷的组合，并且模块的输入的个数与乘除符号的个数相等，这时模块的图标在每一个输入端口附近显示适当的乘除符号，模块的输出是所有标以“×”的输入的乘积，除以所有标有“÷”的输入得到的结果.

如果输入数参数的值是一个大于 1 的标量，模块将所有的输入相乘. 如果所有的输入都是向量，模块的输出是所有输入之间元素对元素的乘法. 如果所有输入是标量，输出也是标量. 对于一个有 n 个输入的模块，如果所有的输入都是向量，输出的每一个元素由式(7.20)产生.

$$y_i = u1_i \times u2_i \times \cdots \times un_i \tag{7.20}$$

如果输入数参数的值等于 1，模块的输出是输入向量的所有元素的标量积：

$$y = \Pi u_i \tag{7.21}$$

如果需要，Sinulink 改变模块图标的大小以显示所有的输入端口. 如果输入的个数改变了，会从模块图标的下部增加或者删去端口.

(3) 模块数据类型

该模块接受任何类型的实数或复数值信号. 所有输入信号必须数据类型一致. 输出与输入数据类型一样.

(4) 模块参数对话框

该模块的参数对话框如图 7.66 所示.

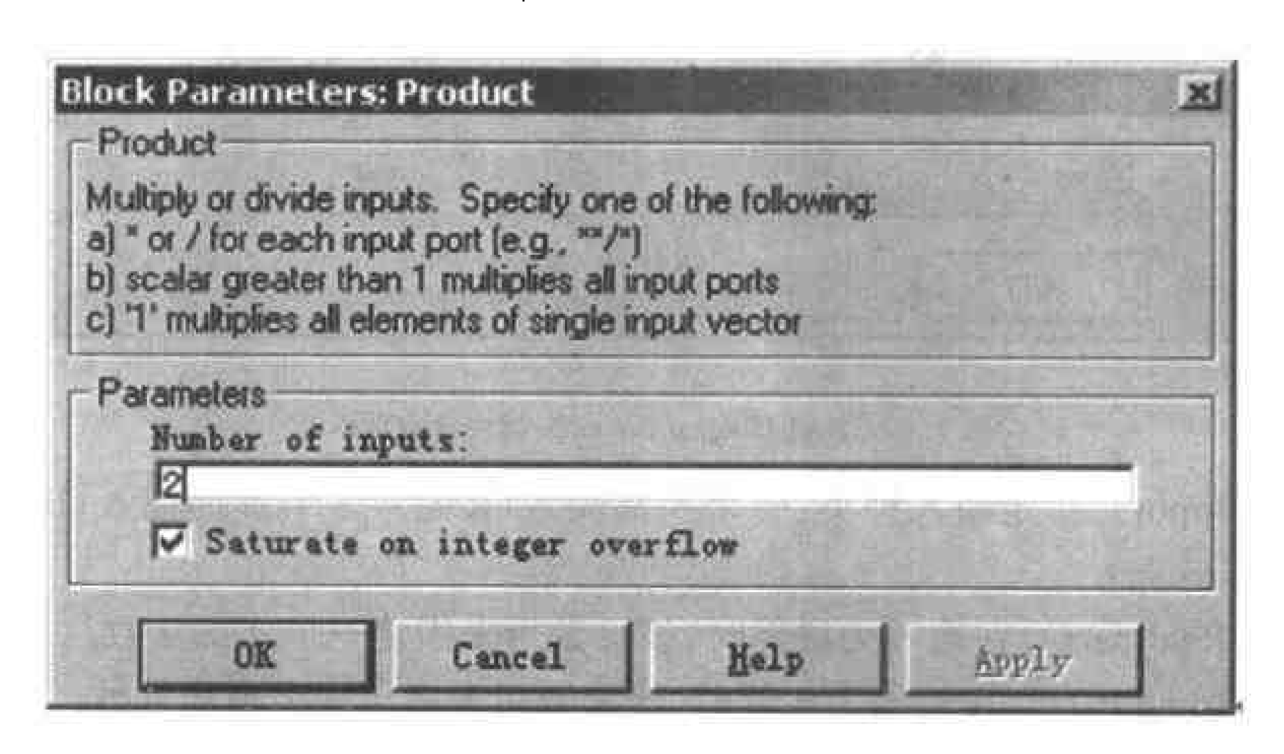


图 7.66 Product 模块的对话框

1) 输入数(Number of inputs),输入的个数或乘除符合的个数. 缺省值为 2.

2) 整数溢出饱和(Saturate on integer overflow),当模块产生整数溢出时,输出饱和值.

(5) 模块特点

1) 直接馈通;

2) 采样时间由驱动模块继承;

3) 可以标量扩展;

4) 可向量化;

5) 没有过零区间.

7.5.14 Real-Imag to Complex

(1) 模块功能

转换实部和虚部为复数信号.

(2) 模块说明

该模块将实部和(或)虚部输入转换为复数值输出信号. 输入可以是大小一样的向量,或者一个输入为向量,另一个为标量. 如果输入为向量,则输出为复数信号向量. 实部输入向量元素映射相应复数输出元素的实部;虚部输入向量元素同样映射相应复数输出元素的虚部. 标量输入映射所有复数输出信号的相应部分(实部或虚部).

(3) 模块数据类型

该模块输入必须是双精度类型实数值信号,输出为双精度类型复数值信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.67 所示.

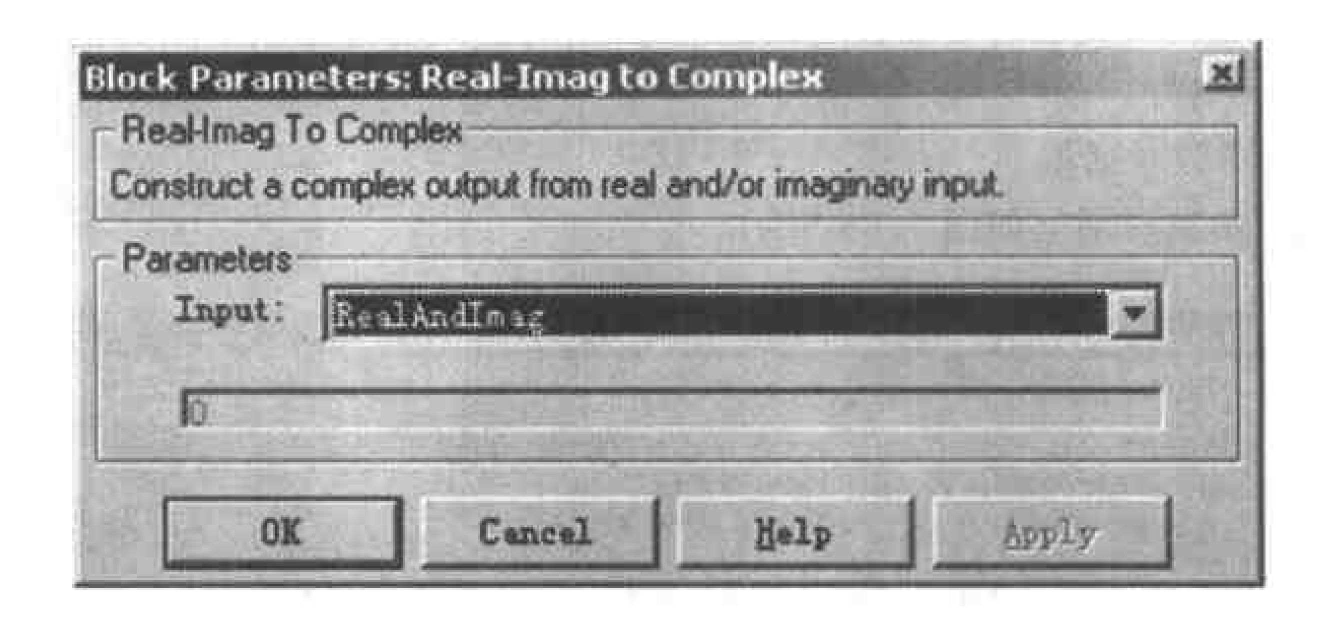


图 7.67　Real-Imag to Complex 模块参数对话框

1）输入(Input)，指定输入种类：一个实部输入(Real)，一个虚部输入(Imag)，或者都输入(RealAndImag).

2）实部(Real part)，在输入为虚部时，该参数用来指定一个实部常数.

3）虚部(Imag part)，在输入为实部时，该参数用来指定一个虚部常数.

(5) 模块特点

1）直接馈通；

2）采样时间由驱动模块继承；

3）当需要两个输入时，输入可以标量扩展；

4）可向量化；

5）没有过零区间.

7.5.15　Relational Operator(关系运算)

(1) 模块功能

对输入执行指定的关系运算.

(2) 模块说明

Relational Operator 模块对它的两个输入执行关系运算并且根据表 7.10 产生输出.

表 7.10　关系运算符

运算符	输　出
==	如果第一个输入等于第二个输入时为真(TRUE)
!=	如果第一个输入不等于第二个输入时为真(TRUE)
<	如果第一个输入小于第二个输入时为真(TRUE)
<=	如果第一个输入小于或等于第二个输入时为真(TRUE)
>=	如果第一个输入大于或等于第二个输入时为真(TRUE)
>	如果第一个输入大于第二个输入时为真(TRUE)

如果结果是真(TRUE)，输出是 1；如果是假(FALSE)，输出是 0. 可以指定输入为标量、向量或者标量与向量的组合：

输入为标量时，输出也是标量；输入是向量时，输出也是向量，并且它的每一个元素是

两个输入向量对应元素比较的结果；如果输入是标量与向量的组合，输出是一个向量，它的每一个元素是标量输入与向量输入的对应元素比较的结果.

模块的图标显示了被选取的关系运算符.

(3) 模块数据类型

该模块接受任何类型的实数信号. 其输出为一逻辑类型的信号；在逻辑兼容模式激活时，输出为双精度类型信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.68 所示.

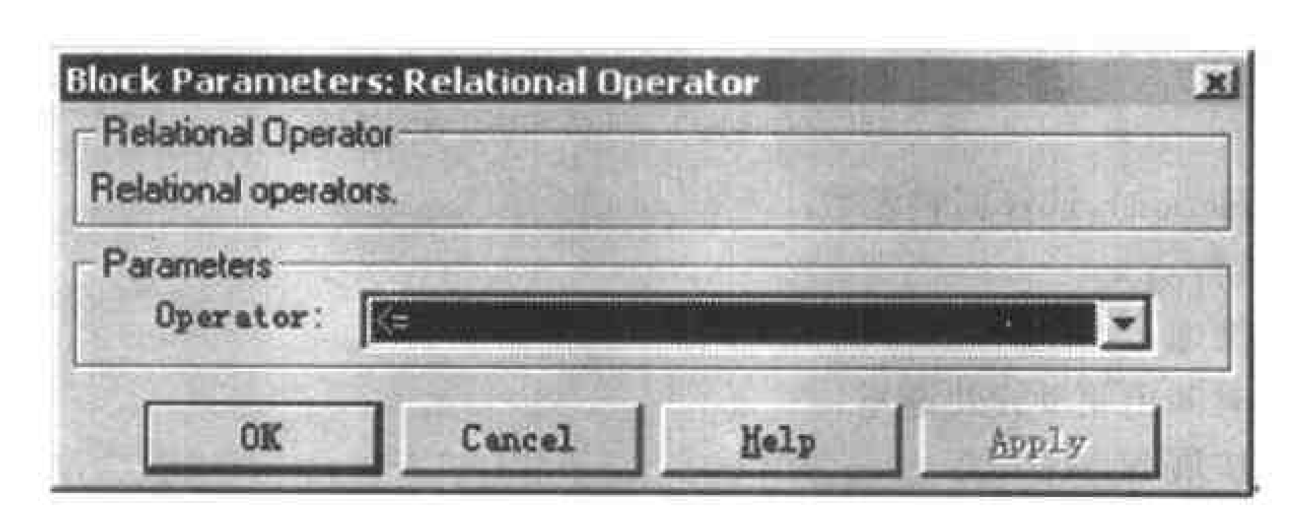


图 7.68　Relational Operator 模块参数对话框

运算符(Operator)，该参数用来选择用于模块输入的关系运算符.

(5) 模块特点

1) 直接馈通；

2) 采样时间由驱动模块继承；

3) 输入可以标量扩展；

4) 可向量化；

5) 有过零区间，检测输出变化.

7.5.16　Rounding Function(圆整函数)

(1) 模块功能

执行圆整函数.

(2) 模块说明

Rounding Function 模块执行普通的数学圆整函数.

可以从 Function 列表中选择 floor，ceil，round 和 fix 这些函数中的一个. 模块的输出是对输入执行相应函数的结果.

函数的名字显示在模块图标上.

如果想对输出进行向量化，应使用 Rounding Function 模块而不是 Fcn 模块，因为 Fcn 模块只能生成标量输出.

(3) 模块数据类型

该模块接受和输出双精度类型实数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.69 所示.

Block Parameters: Rounding Function
Rounding
Rounding operations.
Parameters
Function: floor
OK Cancel Help Apply

图 7.69 Rounding Function 模块参数对话框

函数(Function),选择圆整函数.

(5) 模块特点

1) 直接馈通;

2) 采样时间由驱动模块继承;

3) 标量扩展 N/A;

4) 可向量化;

5) 没有过零区间.

7.5.17 Sign(符号)

(1) 模块功能

显示输入的正负号.

(2) 模块说明

Sign 模块显示输入的符号:

当输入大于 0 时输出为 1;当输入等于 0 时输出为 0;当输入小于 0 时输出为 −1.

(3) 模块数据类型

该模块接受和输出双精度类型实数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.70 所示.

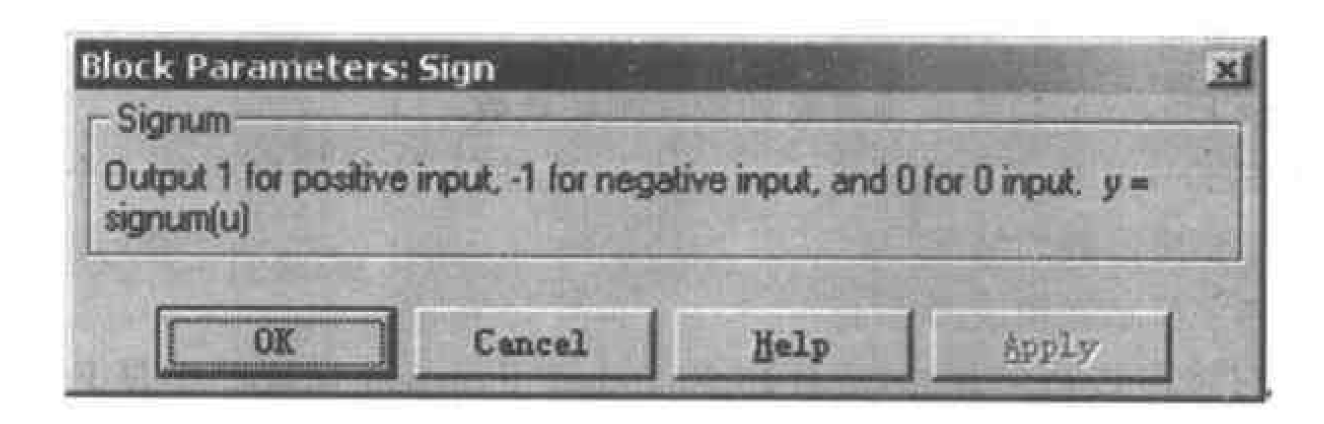


图 7.70 Sign 模块参数对话框

(5) 模块特点

1) 直接馈通;

2）采样时间由驱动模块继承；

3）标量扩展 N/A；

4）可向量化；

5）有过零区间，检测何时经过 0 点.

7.5.18　Slider Gain(滑块增益)

（1）模块功能

使用滑块改变标量增益.

（2）模块说明

通过 Slider Gain 模块可以在仿真期间使用滑块改变标量增益. 该模块接收一个输入并且产生一个输出.

（3）模块数据类型

该模块接受除逻辑类型外的任何类型实数或复数值标量、向量，输出与输入类型相同. 输入向量元素必须类型相同. 增益参数可以是任何类型的实数或复数值标量.

（4）模块参数对话框

该模块的参数对话框如图 7.71 所示.

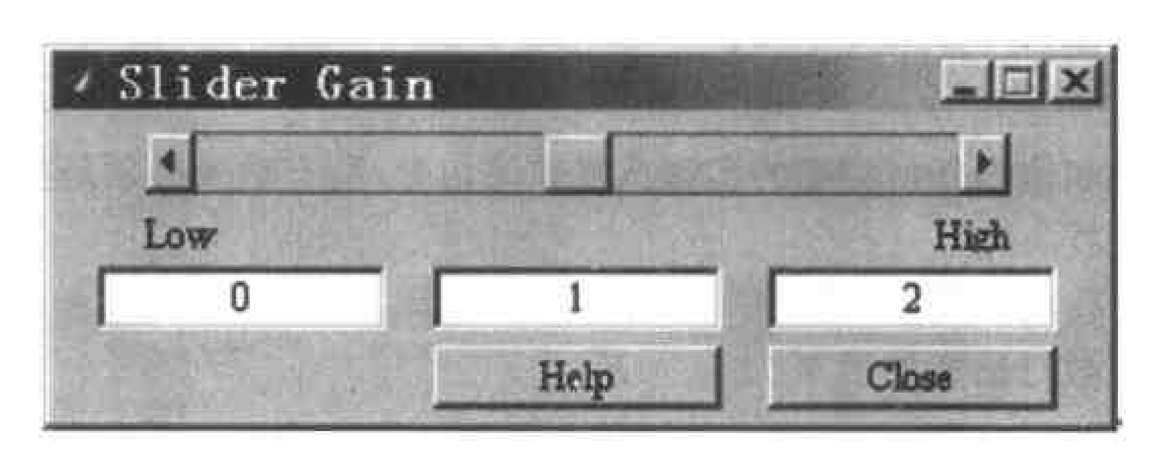


图 7.71　Slider Gain 模块参数对话框

1）下限(Low)，滑块范围的下限. 缺省值为 0.

2）上限(High)，滑块范围的上限. 缺省值为 2.

该模块的对话框中的编辑域分别(从左到右)是下限值、当前值和上限值. 可以通过两种方法改变增益：通过操作滑块，或者通过在对话框的当前值域中输入新的数值. 可以通过改变上下限的值改变增益的范围. 通过点击 Close 按钮关闭对话框.

如果点击滑块的左箭头或者右箭头，当前值减小或者增加滑块范围的 1%. 如果点击滑块指针任何一边的条形区域，当前值改变滑块范围的 10%.

要使用向量增益应考虑用 Gain 模块，要使用矩阵增益应考虑用 Matrix Gain 模块.

（5）模块特点

1）直接馈通；

2）采样时间由驱动模块继承；

3）增益可以标量扩展；

4）状态 0；

5）可向量化；

6）没有过零区间.

7.5.19 Sum(和)

(1) 模块功能

产生各个输入之和.

(2) 模块说明

Sum 模块将各个标量和(或者)向量输入相加，或者当输入只有一个向量时将它的所有元素相加，这取决于输入模块的数目.

如果该模块有多个输入，它将各个输入之间元素对元素相加的和作为其输出. 如果所有输入是标量，输出也是标量. 如果该模块有 n 个输入，并且每个输入都是向量，输出的每个元素由下式得到：

$$y_i = u1_i + u2_i + \cdots + un_i \tag{7.22}$$

如果模块只有一个向量输入，模块的输出是一个标量，它是输入的所有元素之和：

$$y = \Sigma u_i \tag{7.23}$$

Sum 模块在它图标上的端口附近显示合适的正负号，并且重画它的端口以与在 List of signs 参数中指定的符号的数目相匹配. 如果有必要，Simulink 改变模块图标的大小以显示所有的输入端口.

(3) 模块数据类型

该模块接受和输出任何类型的实数或复数值信号. 所有输入必须是类型一样，输出与输入数据类型一样.

(4) 模块参数对话框

该模块的参数对话框如图 7.72 所示.

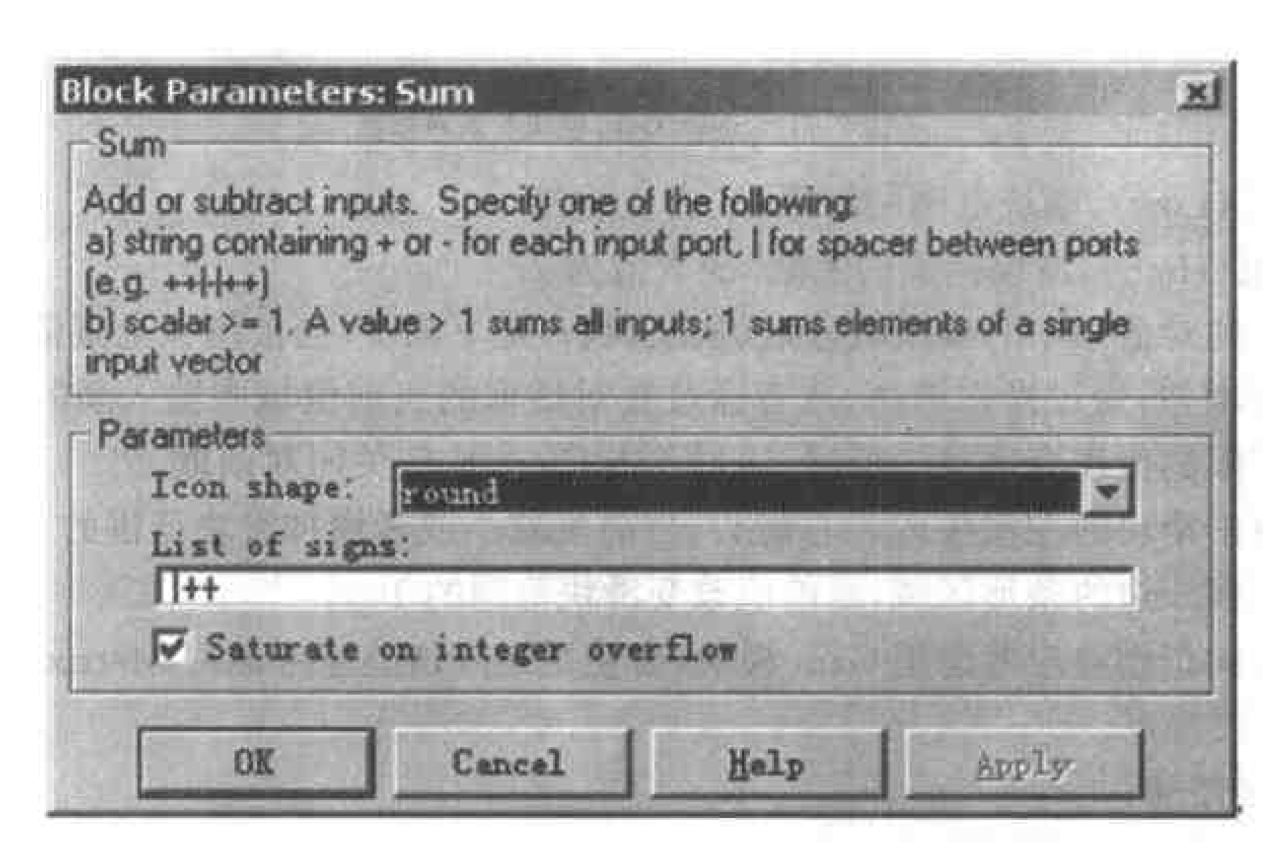


图 7.72 Sum 模块参数对话框

1）图标形状(Icon shape)，从该下拉框中选择圆形或矩形图标形状.

2）符号列表(List of signs)，该参数可以是一个常数或“＋”、“－”、“|”符合组合. 指定一个常数，表示输入的个数，将重画图标；“|”将在相应的位置产生一个空位.

3）整数溢出饱和(Saturate on integer overflow)，如果选择，在整数溢出时，将输出饱和值.

(5) 模块特点

1）直接馈通；

2）采样时间由驱动模块继承；

3）可以标量扩展；

4）状态 0；

5）可向量化；

6）没有过零区间.

7.5.20 Trigonometric Function(三角函数)

(1) 模块功能

执行三角函数.

(2) 模块说明

Trigonometric Function 模块执行一些普通三角函数.

可以从 Function 列中选择如下的这些函数之一：sin，cos，tan，asin，acos，atan，atan2，sinh，cosh 和 tanh. 该模块的输出是对输入执行函数的结果.

函数的名字显示在模块的图标中. Simulink 自动地画出适当数目的输入端口.

如果想得到向量化的输出应使用 Trigonometric Function 模块而不是 Fcn 模块，因为 Fcn 模块只能生成标量输出.

(3) 模块数据类型

该模块接受和输出双精度类型实数或复数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.73 所示.

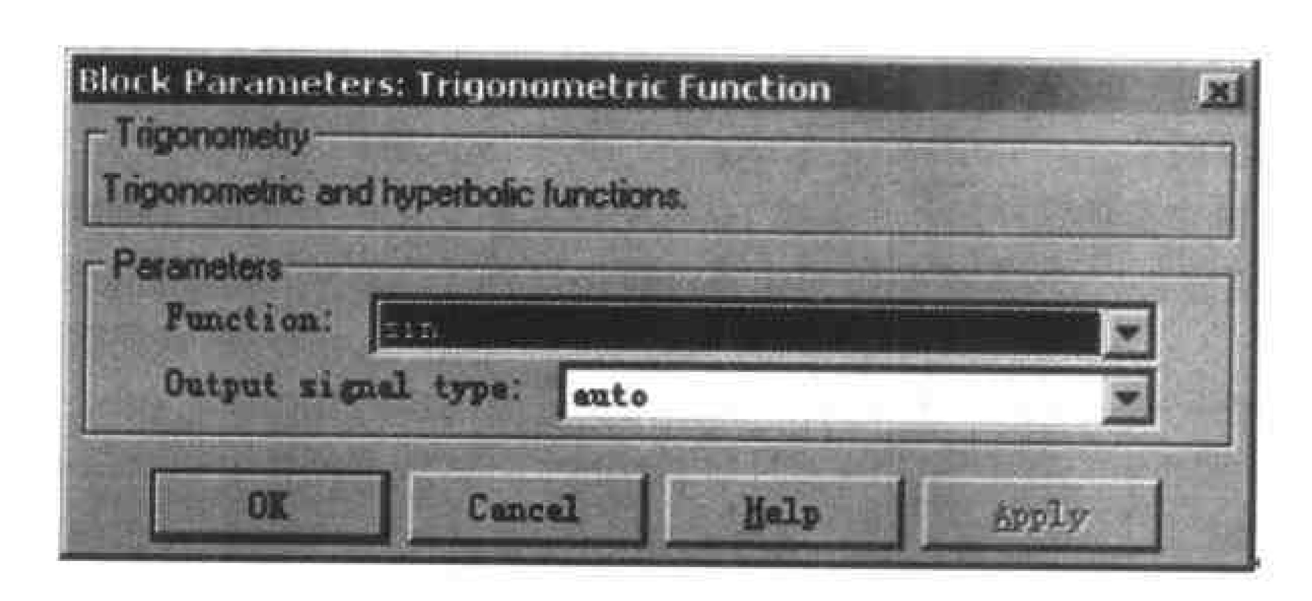


图 7.73 Trigonometric Function 模块的对话框

1）函数(Function)，选择要执行的三角函数.

2）输出信号类型(Output signal type)，指定输出为实数(real)、复数(complex)或自动(auto).

(5) 模块特点

1）直接馈通；

2）采样时间由驱动模块继承；

3）在需要两个输入时，输入可以标量扩展；

4）可向量化；

5）没有过零区间.

7.6　Nonlinear 库中的模块

Nonlinear 库中包含的模块如表 7.11 所列，各个模块的图标如图 6.7 所示.

表 7.11　Nolinear 库中的各模块

模　块　名	功　　能
Backlash	模拟有间隙系统的行为
Coulomb & Viscous Friction	模拟在零点处不连续，在其它地方有线性增益的系统
Dead Zone	提供输出为零的区域
Manual Switch	在两个输入之间切换
Multiport Switch	在模块的多个输入之间切换
Quantizer	以指定的间隔离散化输入
Rate Limiter	限制信号的变化速度
Relay	在两个常数中选出一个作为输出
Saturation	限制信号的变化范围
Switch	在两个输入之间切换

7.6.1　Backlash 模块

（1）模块功能

对有间隙系统的行为进行模拟.

（2）模块说明

Backlash 模块实现这样的一个系统：系统中输入的变化会引起输出有相同的变化. 然而，当输入改变方向后，输入的初始变化将对输出没有什么影响. 系统中间隙的量被称为死带. 死带大约以输出为中心. 缺省死带宽度为 1，并且初始输出为 0 的模块的初始状态如图 7.74 所示.

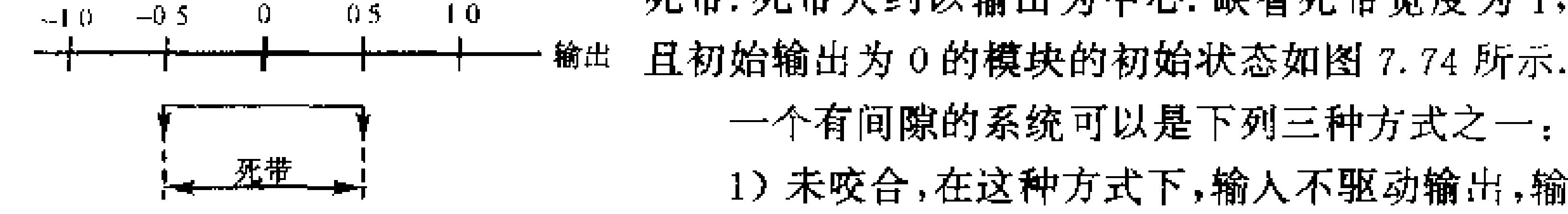


图 7.74　Backlash 模块的初始状态

一个有间隙的系统可以是下列三种方式之一：

1）未咬合，在这种方式下，输入不驱动输出，输出保持为常数.

2）正向咬合，在这种方式下，输入是递增的（斜率为正），输出等于输入减去死带宽度的一半.

3）负向咬合，在这种方式下，输入是递减的（斜率为正），并且输出等于输入加上死带宽度的一半.

如果初始输入是在死带的范围之外，初始输出参数值将根据模块在仿真开始时是正

向还是负向来确定其值是输入加上死带宽度的一半还是减去死带宽度的一半.

例如,Backlash 模块可以用于模拟两个齿轮的啮合.输入和输出都是在末端有一个齿轮的轴,并且输出轴由输入轴驱动.轮齿之间多余的空间引入了间隙,空间的宽度就是参数死带宽度 Deadband width 的值.如果系统初始时未咬合,输出(被驱动齿轮的位置)由初始输出参数 Initial output 确定.

(3) 模块数据类型

该模块接受和输出双精度类型的实数值信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.75 所示.

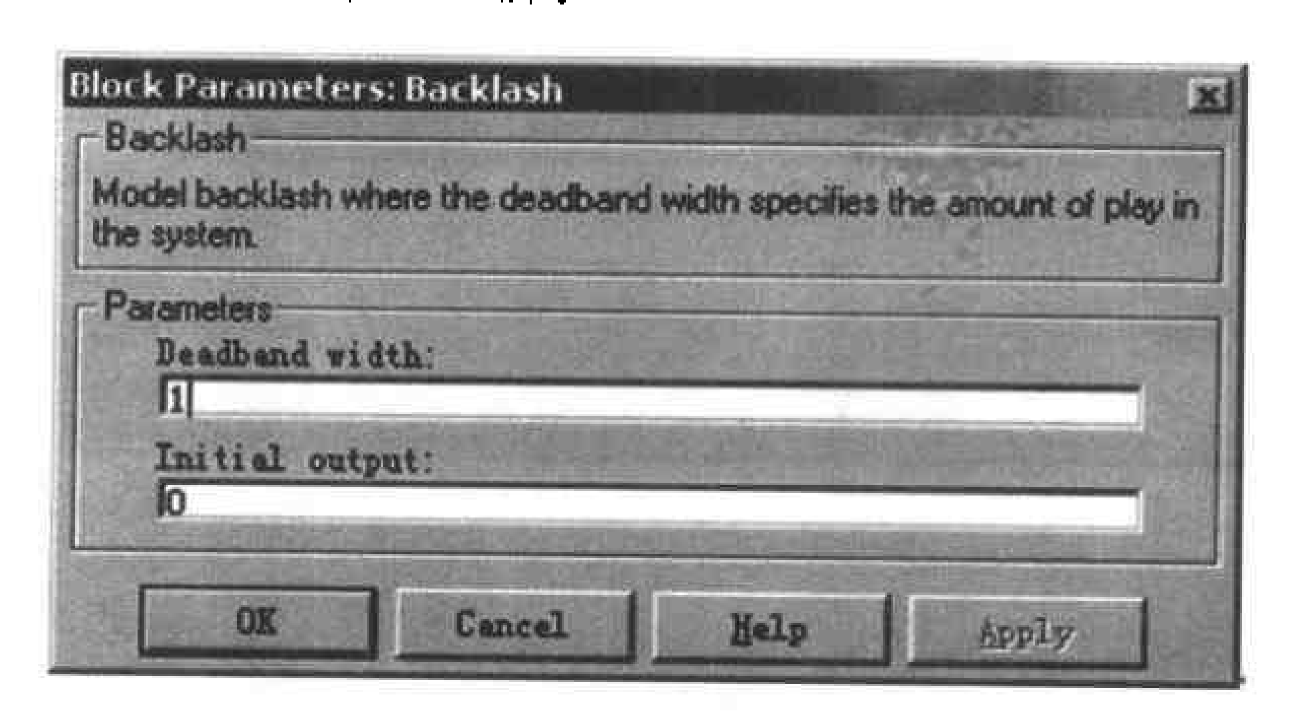


图 7.75 Backlash 模块参数对话框

1) 死带宽度(Deadband width),指死带的宽度.缺省值为 1.

2) 初始输出(Initial output),初始的输出值.缺省值为 0.

(5) 模块特点

1) 直接馈通;

2) 采样时间由驱动模块继承;

3) 可以标量扩展;

4) 可向量化;

5) 有过零区间,用上下阈值检测接触咬合.

7.6.2 Coulomb and Viscous Friction(库仑和粘性摩擦)

(1) 模块功能

对在 0 处不连续在其它的地方有线性增益的函数的建模.

(2) 模块说明

Coulomb and Viscous Friction 模块对库仑(静态)和粘性(动态)摩擦进行建模.该模块模拟在 0 处不连续,在其它的地方有线性增益的函数.Offset(偏移量)对应于库仑摩擦,Gain(增益)对应于粘性摩擦.实现模块的表达式是:

y=sign(u) * (Gain * abs(u)+Offset)

其中 y 为输出,u 是输入,Gain 和 Offset 是模块的参数.

模块有一个输入和一个输出.

(3) 模块数据类型

该模块接受和输出双精度类型的实数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.76 所示.

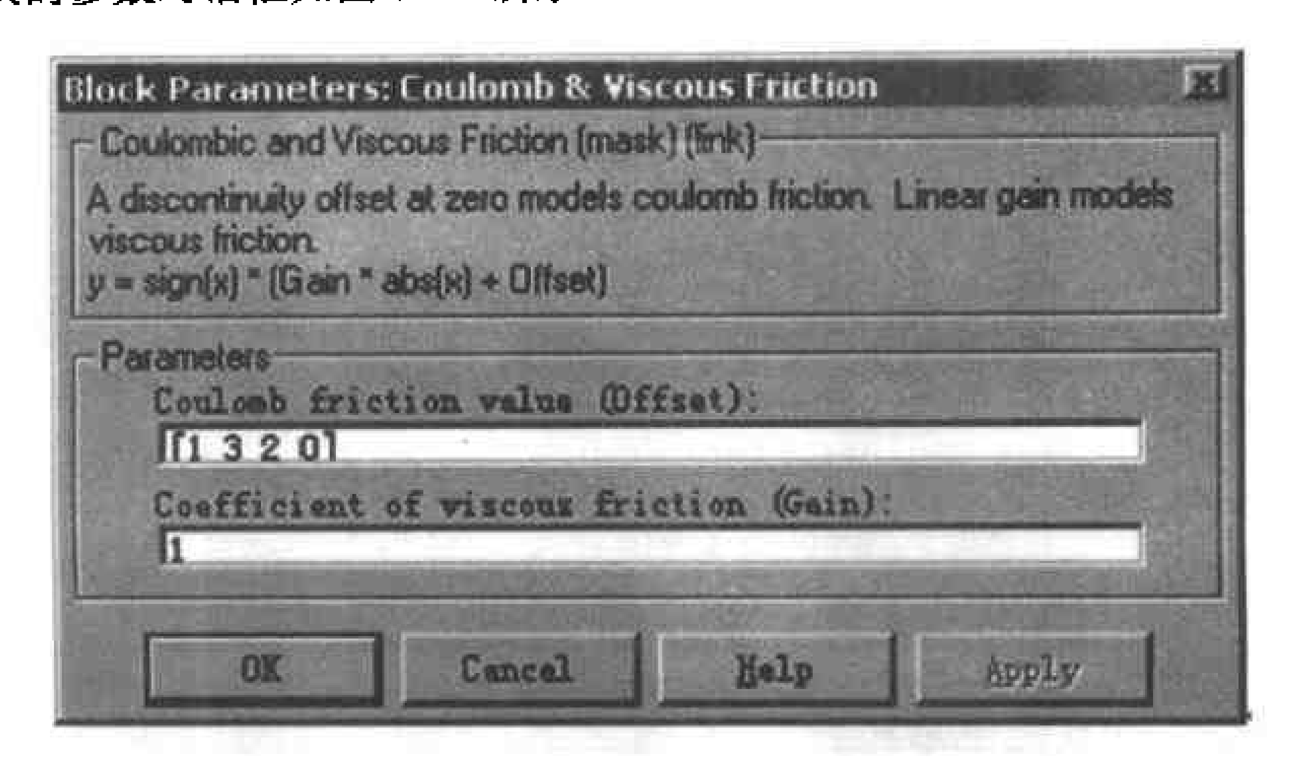


图 7.76　Coulomb and Viscous Friction 模块参数对话框

1) 库仑摩擦值(Coulomb friction value),对于所有输入值的偏移量. 缺省值为:[1 3 2 0].

2) 粘性摩擦系数(Coefficient of viscous friction),在非 0 输入点的信号增益. 缺省值为 1.

(5) 模块特点

1) 直接馈通;

2) 采样时间由驱动模块继承;

3) 没有标量扩展;

4) 可向量化;

5) 有过零区间,克服静态摩擦的点.

7.6.3　Dead Zone(死区)

(1) 模块功能

提供一个输出为 0 的区域.

(2) 模块说明

Dead Zone 模块在指定的区域内生成 0 输出,这一区域也叫死区. 死区的下限和上限由参数死区开始(Start of dead zone)和死区结束(End of dead zone)指定. 模块的输出取决于输入和死区.

1) 如果输入在死区内(大于下限并且小于上限),则输出为 0.

2) 如果输入大于或者等于上限,输出是输入减去上限.

3) 如果输入小于或者等于下限,输出是输入减去下限.

例 7.4 该例的输入为一正弦波，死区的上下限分别为+0.2和-0.6.模型图如图7.77所示.

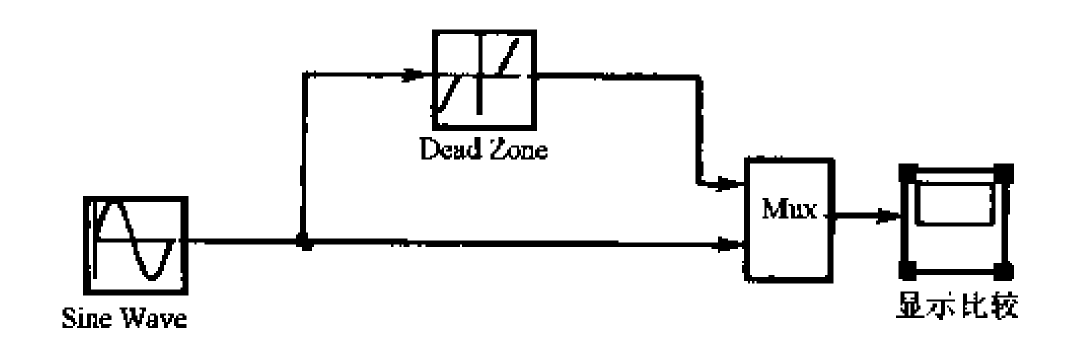


图 7.77 包含有 Dead Zone 模块的示例模型

图 7.78 显示了 Dead Zone 模块对正弦波的作用.当输入在-0.6和+0.2之间时，输出为0.

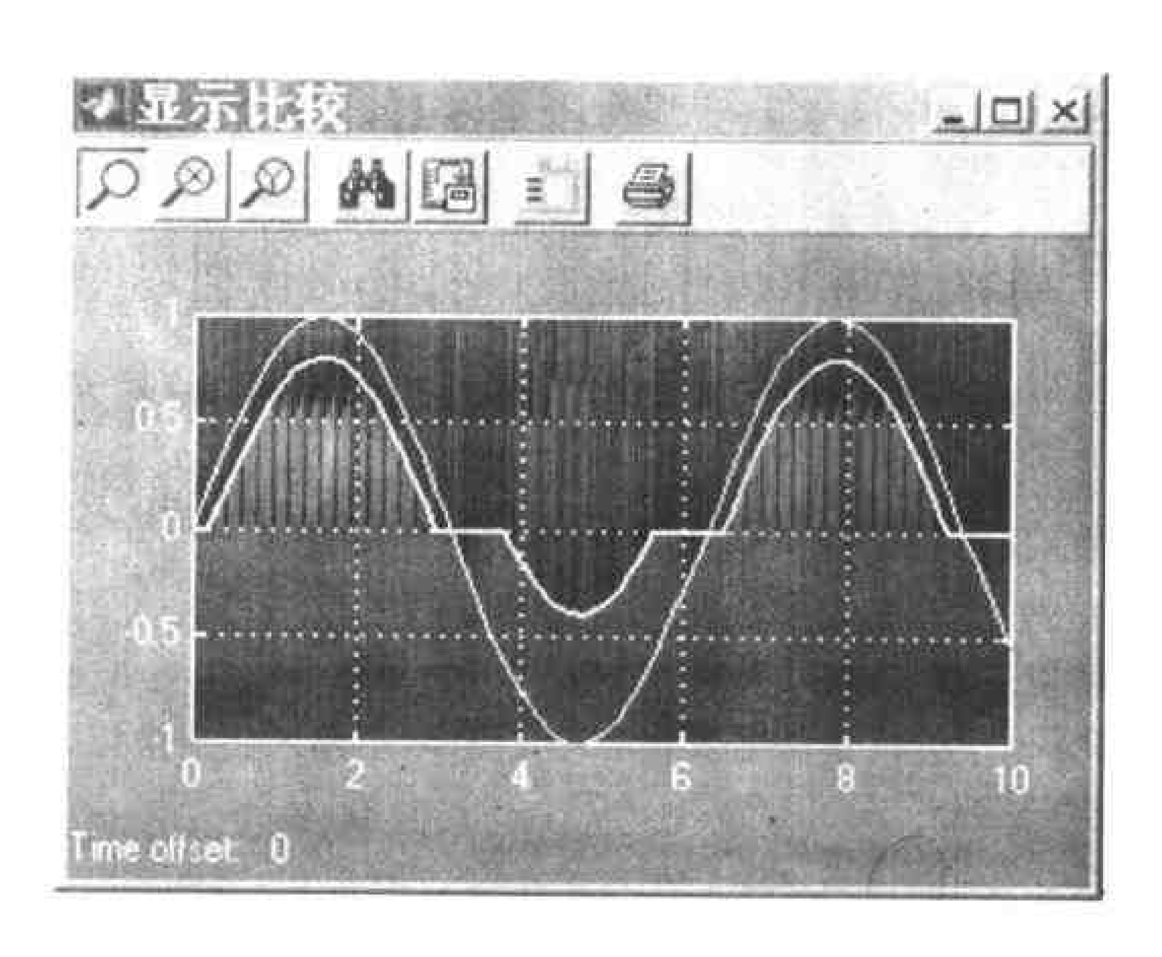


图 7.78 示例模型输出图形

(3) 模块数据类型

该模块接受和输出双精度类型的实数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.79 所示.

1) 死区开始(Start of dead zone)，死区下限.缺省值为-0.5.

2) 死区结束(End of dead zone)，死区上限.缺省值为 0.5.

(5) 模块特点

1) 直接馈通；

2) 采样时间由驱动模块继承；

3) 参数可以标量扩展；

4) 可向量化；

5) 有过零区间，检测死区限.

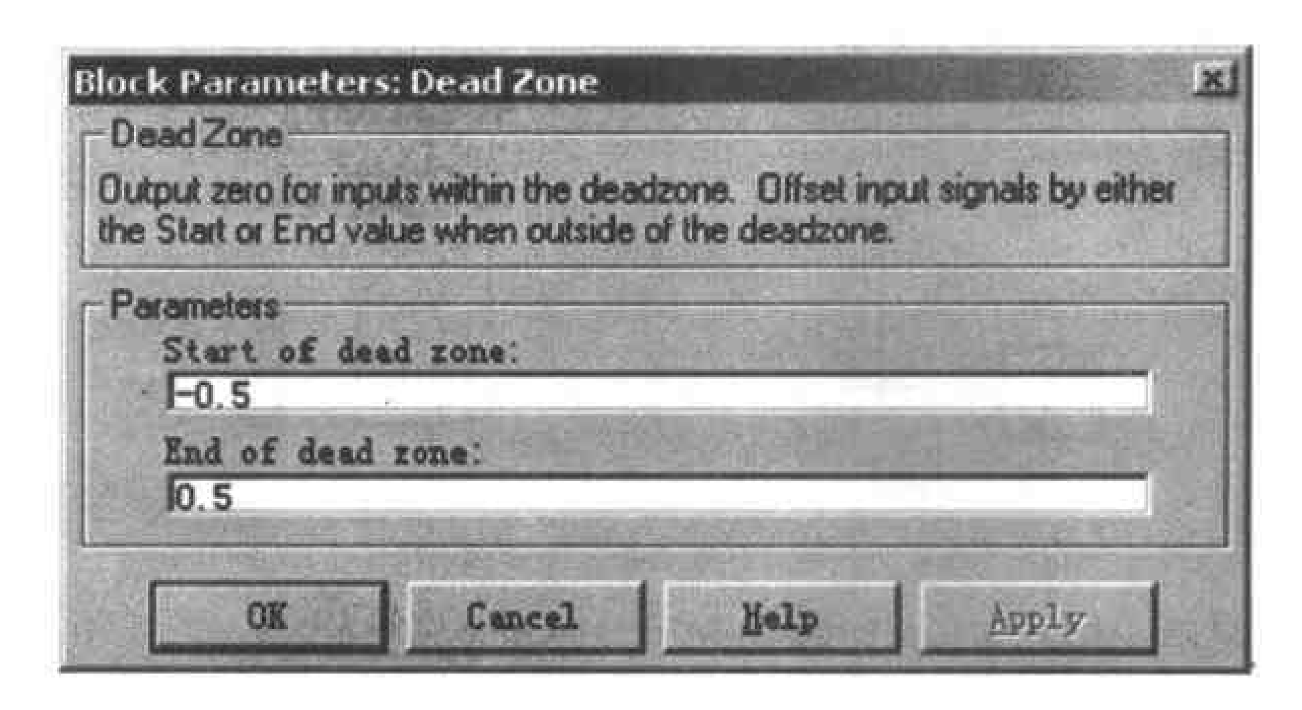


图 7.79　Dead Zone 模块参数对话框

7.6.4　Manual Switch(手动开关)

(1) 模块功能

在两个输入之间进行切换.

(2) 模块说明

Manual Switch 模块是一个在它的两个输入之间选择其一传给输出的拨动开关. 要在两个输入之间进行拨动,只用双击它的图标即可(它没有对话框). 被选取的输入传给输出,而没有被选取的输入被丢弃了. 可以在仿真开始之前就设置好开关,当仿真正在运行的时候,可以切换开关以控制信号的流向.

(3) 模块数据类型

Manual Switch 模块接收任何类型的输入类型. 两个输入必须是相同的数值或数据类型. 输出与输入类型一致.

(4) 模块特点

1) 直接馈通;

2) 采样时间由驱动模块继承;

3) 标量扩展 N/A;

4) 可向量化;

5) 没有过零区间.

7.6.5　Multiport Switch(多路转换开关)

(1) 模块功能

从模块的多个输入中选择一个作为输出.

(2) 模块说明

Multiport Switch 模块从模块的多个输入中选择一个作为输出.

第一个(最顶端)输入是控制端,其它的输入是数据输入端. 控制端的值确定了哪一个数据输入端传送给输出端. Simulink 将控制端输入值圆整为最为接近的正整数,并且切换到与这一数相对应的数据输入端,如表 7.12 所示.

表 7.12　多路开关控制端值

控制端的值	切换到的转换端
小于 1	超过范围错误
从 1 至 1.4999…	第一个
从 1.5 到 2.4999…	第二个
从 2.5 到 3.4999…	第三个
…	…
大于数据输入端口数	超过范围错误

数据输入端可以是标量也可以是向量，同样，控制输入端可以是标量也可以是向量. 模块的输出由如下的规则确定：

1）如果输入是标量，输出也为标量.

2）如果模块有多个数据输入端，并且至少有一个输入是向量，输出也为向量. 任何标量输入被扩展为向量.

3）如果模块只有一个数据输入端，并且该输入端是向量，模块的输出是向量中对应于控制输入端圆整值的元素.

(3) 模块数据类型

该模块控制输入接受除逻辑类型外的任何内置数据类型实数值信号. 数据输入端接受任何类型的实数或复数值输入. 所有数据输入必须是类型一样，输出与输入数据类型一样.

(4) 模块参数对话框

该模块的参数对话框如图 7.80 所示.

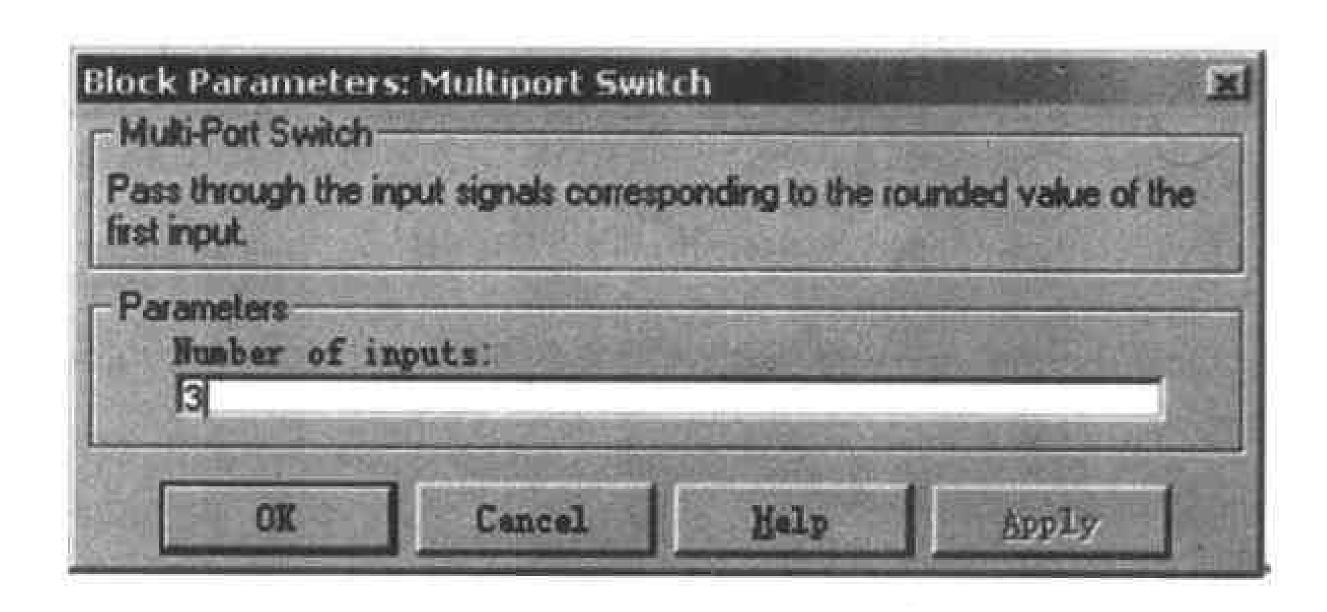


图 7.80　Multiport Switch 模块参数对话框

输入数(Number of inputs)，指模块的数据输入个数.

(5) 模块特点

1）直接馈通；

2）采样时间由驱动模块继承；

3）可以标量扩展；

4）可向量化；

5）没有过零区间.

7.6.6　Quantizer（量化）

（1）模块功能

以指定的间隔对输入进行离散化.

（2）模块说明

Quantizer 模块将其输入信号传给阶梯函数，以使输入轴上连续的一段区间映射为输出轴上的一点. 它的效果是将一个光滑的信号量化成阶跃的输出. 输出是通过使用圆整为最邻近的点的方法得到的，它产生的结果是关于零点对称的：

$$y = q * \mathrm{round}(u/q)$$

其中 y 是输出，u 是输入，q 是 Quantization interval 参数.

（3）模块数据类型

该模块接受和输出双精度类型信号.

（4）模块参数对话框

该模块的参数对话框如图 7.81 所示.

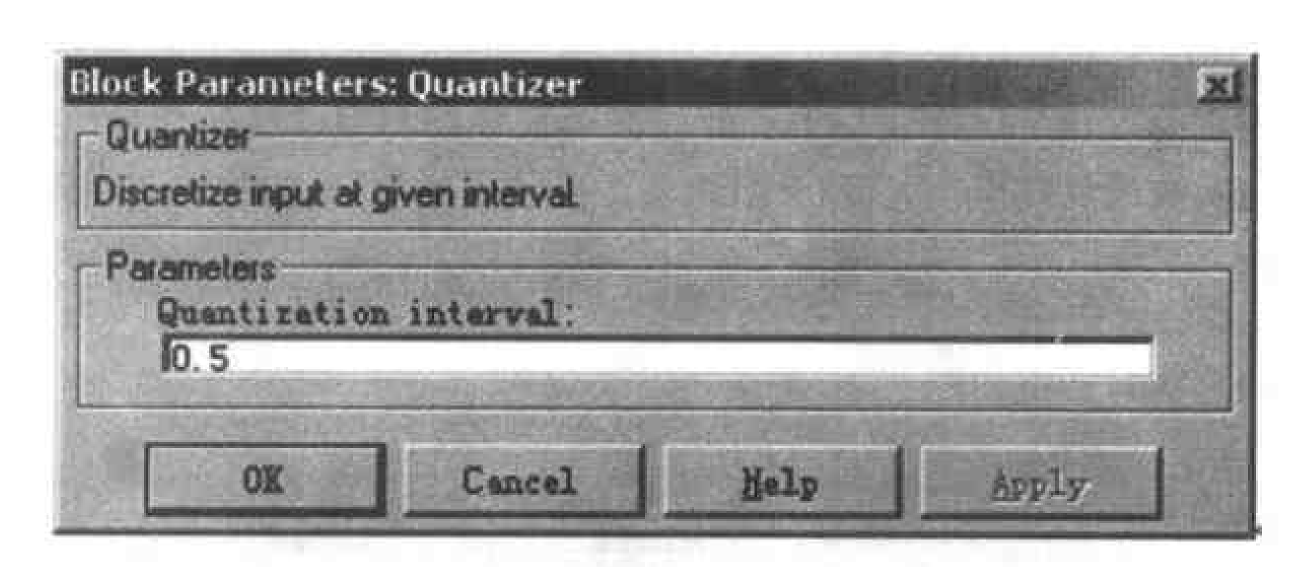


图 7.81　Quantizer 模块参数对话框

量化间隔(Quantization interval)，输出被量化的间隔，允许输出值为 n * q，n 为整数，q 即量化间隔. 缺省值为 0.5.

（5）模块特点

1）直接馈通；

2）采样时间由驱动模块继承；

3）参数可以标量扩展；

4）可向量化；

5）没有过零区间.

7.6.7　Rate Limiter（限速器）

（1）模块功能

限制信号改变的速率.

(2) 模块说明

Rate Limiter 模块限制通过它的信号的一阶导数.输出不能比给定的限改变得更快.导数用式(7.24)的方程计算.

$$\text{rate} = \frac{u(i) - y(i-1)}{t(i) - t(i-1)} \tag{7.24}$$

$u(i)$和 $t(i)$是模块当前的输入和时间,$y(i-1)$和 $t(i-1)$是前一步的输出和时间.输出由 rate 与 Rising slew rate 和 Falling slew rate 参数比较的结果确定:

1) 如果 rate 比 Rising slew rate 参数(R)大,输出由式(7.25)确定.

$$y(i) = \Delta t \cdot R + y(i-1) \tag{7.25}$$

2) 如果 rate 比 Falling slew rate 参数(F)小,输出由式(7.26)确定.

$$y(i) = \Delta t \cdot F + y(i-1) \tag{7.26}$$

3) 如果 rate 是在 Rising slew rate 和 Falling slew rate 之间,输出的变化等于输入的变化:

$$y(i) = u(i) \tag{7.27}$$

(3) 模块数据类型

该模块接受和输出双精度类型信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.82 所示.

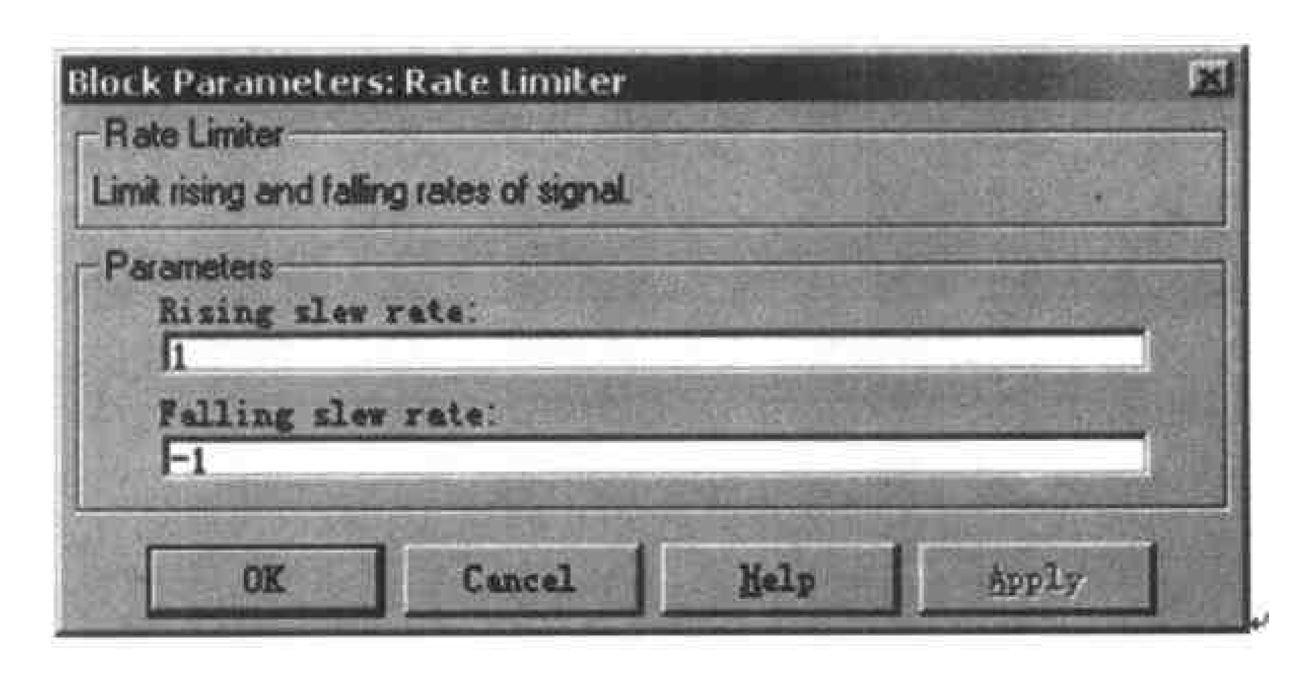


图 7.82　Rate Limiter 模块参数对话框

1) 上升速率(Rising slew rate),指增长的输入信号导数限.

2) 下降速度(Falling slew rate),指减小的输入信号导数的限.

(5) 模块特点

1) 直接馈通;

2) 采样时间由驱动模块继承;

3) 输入参数可以标量扩展;

4) 可向量化;

5) 没有过零区间.

7.6.8 Relay(继电器)

(1) 模块功能

在两个常数之间切换.

(2) 模块说明

Relay 模块允许输出在两个给定的值之间切换. 当继电器开时,它保持为开的状态,直到输入降得比切断点(Switch off point)参数的值低时为止. 当继电器关时,它保持为关的状态,直到输入超过接通点(Switch on point)参数的值时为止. 模块接收一个输入并且产生一个输出.

当指定的接通点(Switch on point)值比切断点(Switch off point)值大时,它可以模拟滞后现象;当指定的接通点(Switch on point)值与切断点(Switch off point)值相等时,模拟以该值为门槛的开关. 接通点的值必须大于或者等于切断点的值.

(3) 模块数据类型

该模块接受和输出双精度类型的实数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.83 所示.

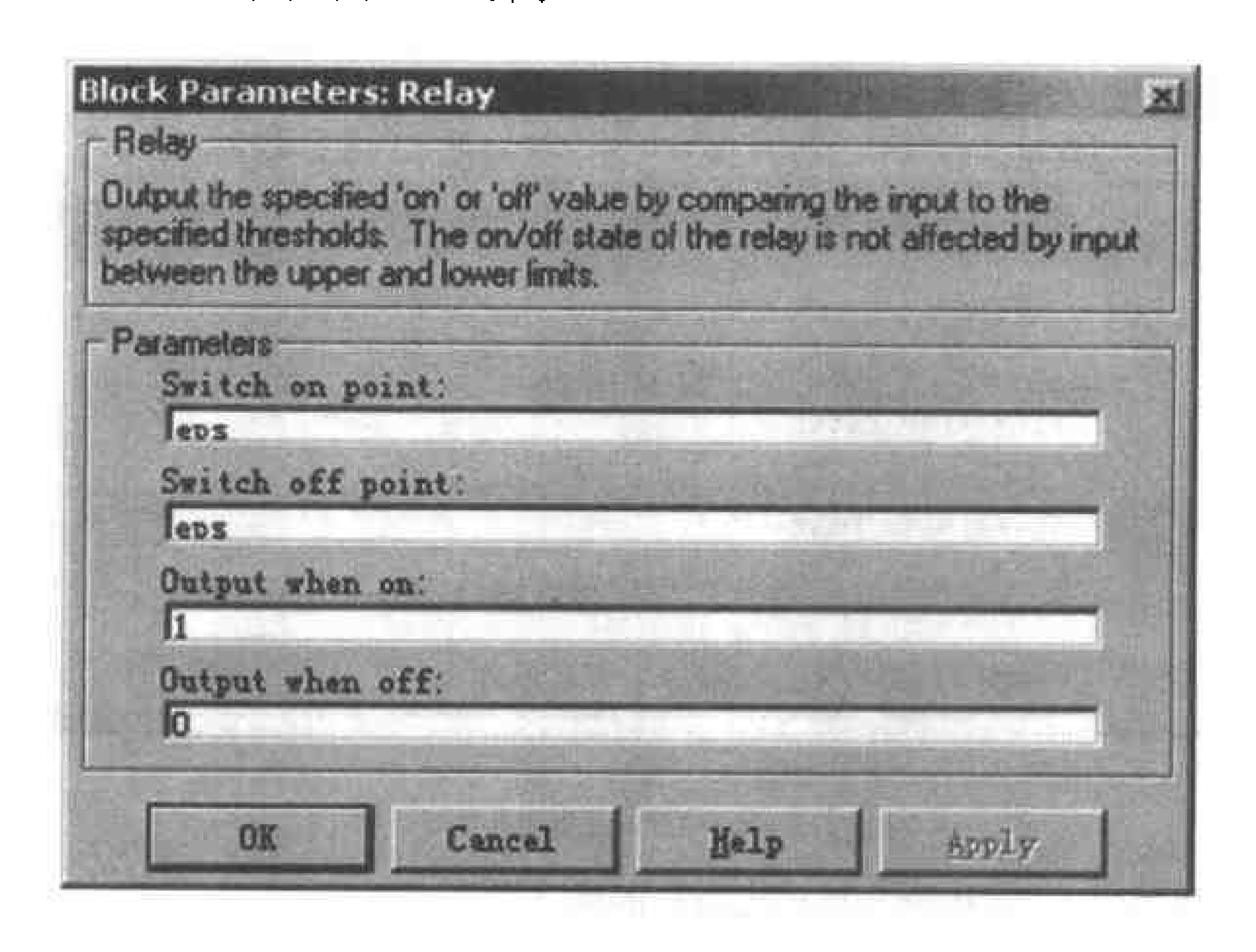


图 7.83 Relay 模块参数对话框

1) 接通点(Switch on point),继电器接通阈值. 缺省值为 eps.

2) 断开点(Switch off point),继电器断开阈值. 缺省值为 eps.

3) 接通时输出(Output when on),继电器接通时输出. 缺省值为 1.

4) 断开时输出(Output when off),继电器断开时输出. 缺省值为 0.

(5) 模块特点

1) 直接馈通;

2）采样时间由驱动模块继承；

3）可以标量扩展；

4）可向量化；

5）有过零区间，检测断开和接通点.

7.6.9 Saturation（饱和）

（1）模块功能

限制信号的范围.

（2）模块说明

Saturation 模块对信号设置上、下边界. 当输入信号在由下限（Lower limit）和上限（Upper limit）参数指定的范围内时，输入信号毫无改变的通过. 当输入信号在边界之外时，信号被削为上边界值或下边界值.

当 Lower limit 和 Upper limit 参数被设为相同的值时，模块输出这一值.

（3）模块数据类型

该模块接受和输出双精度类型的实数信号.

（4）模块参数对话框

该模块的参数对话框如图 7.84 所示.

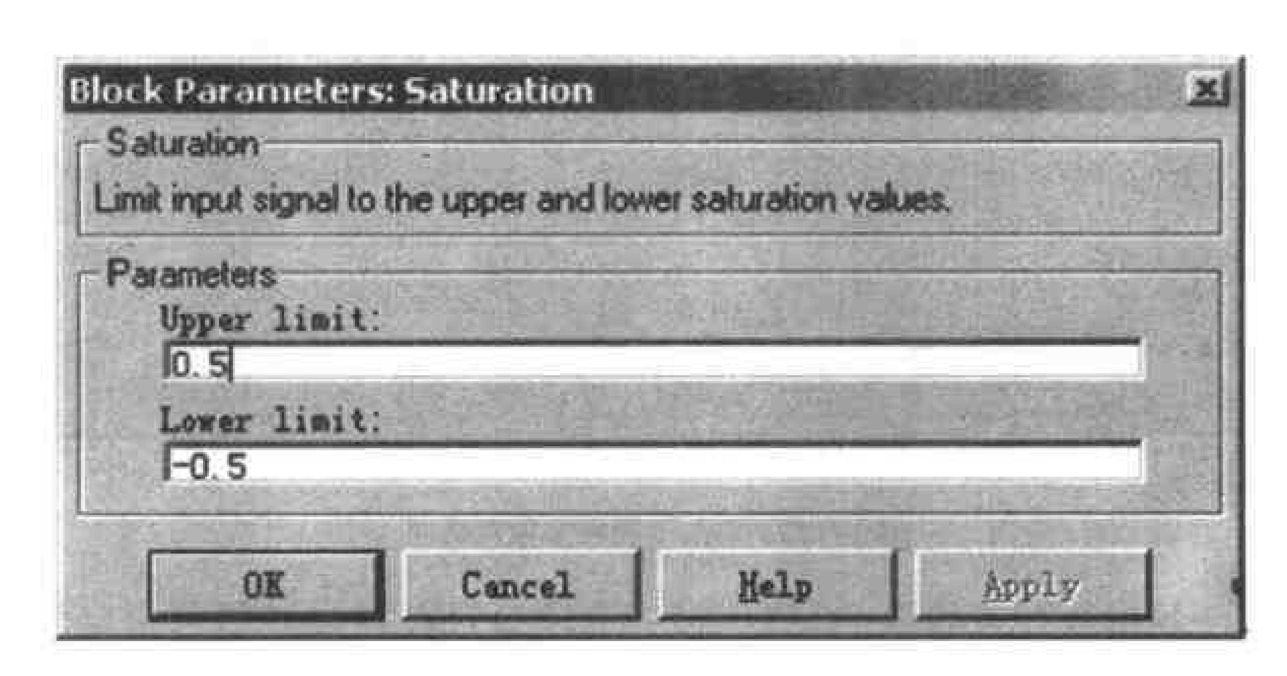


图 7.84 Saturation 模块参数对话框

1）上限（Upper limit），输入信号的上边界，当输入信号超过此值时，模块输出为该值.

2）下限（Lower limit），输入信号的下边界，当输入信号低于此值时，模块输出为该值.

（5）模块特点

1）直接馈通；

2）采样时间由驱动模块继承；

3）参数和输入可以标量扩展；

4）可向量化；

5) 有过零区间,检测何时信号到达极限,何时离散极限.

7.6.10 Switch(选择开关)

(1) 模块功能

选通两个输入中的一个.

(2) 模块说明

Switch 模块有三个输入端口,第二个端口为控制输入端口.根据控制输入的值将它的两个输入中的一个传送给输出.如果控制信号(第二个输入端口)大于或者等于阈值(Threshold)参数值,模块传送第一个输入,否则传送第三个输入.

要用逻辑输入(0 或 1)驱动 Switch 模块,将其 Threshold 设为 0.5.

(3) 模块数据类型

该模块接受任何类型的实数或复数值信号作为开关输入(第一、第三端口).所有输入必须是类型一样,输出与输入数据类型一样.阈值输入必须是逻辑或双精度类型数据.

(4) 模块参数对话框

该模块的参数对话框如图 7.85 所示.

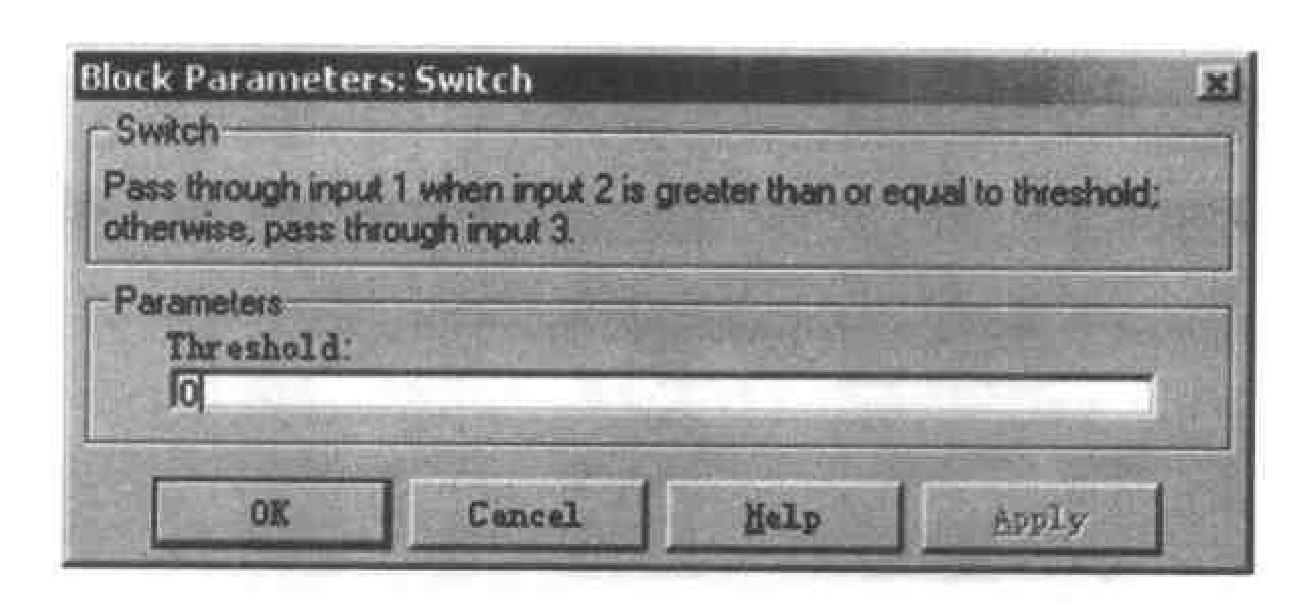


图 7.85 Switch 模块参数对话框

阈值(Threshold),开关转换的控制(第二个端口)值.该参数值可指定为标量或与输入向量宽度一样的向量.

(5) 模块特点

1) 直接馈通;

2) 采样时间由驱动模块继承;

3) 可以标量扩展;

4) 可向量化;

5) 有过零区间,检测开关条件何时发生.

7.7 Signals & Systems 库中的模块

Signals & Systems(信号与系统)库中包含的模块如表 7.13 所列,各个模块的图标如

图 6.9 所示.

表 7.13 Signals & Systems 库中的各模块

模块名	功能
Bus Selector	输出选择的输入信号
Configurable Subsystem	表达从指定库中选择的任何模块
Data Store Memory	定义共享的数据存储区
Data Store Read	从共享数据存储区中读取数据
Data Store Write	写数据到共享数据存储区
Data Type Conversion	转换信号为另一种数据类型
Demux	将向量信号分解为多个输出信号
Enable	给子系统增加一个激活端口
From	从 Goto 模块接收数据
Function-Call Generator	执行函数调用子系统
Goto	将模块的输入传给 From 模块
Goto Tag Visibility	定义 Goto 模块标签的有效范围
Ground	将没有被连接的输入端口接地
Hit Crossing	检查过零点
IC	设定信号的初始值
Inport	为子系统或者外部输入创建一个输入端口
Merge	组合几个输入连线成为一个标量连线
Model Info	显示模型中修改控制信息
Mux	将几条输入连线组合成一条向量连线
Outport	为子系统或者外部输出创建一个输出端口
Probe	输出一个输入信号的宽度、采样时间、和(或)信号类型
Subsystem	表示系统中的一个系统
Terminator	终止没有连接的输出端口
Trigger	给子系统加触发端口
Width	输出输入向量的宽度

7.7.1 Bus Selector

(1) 模块功能

从引入线中选择信号.

(2) 模块说明

Bus Selector 模块从一个 Mux 模块或另外一个 Bus Selector 模块接受信号. 模块有一个输入端口. 输出端口数量取决于 Muxed output 核选框的状态. 如果选中,则信号被组合于输出端口,并且仅有一个输出端口;否则,每个选择信号有一个输出端口.

(3) 模块数据类型

该模块接受和输出任何类型的实数或复数值信号.

（4）模块参数对话框

该模块的参数对话框如图 7.86 所示.

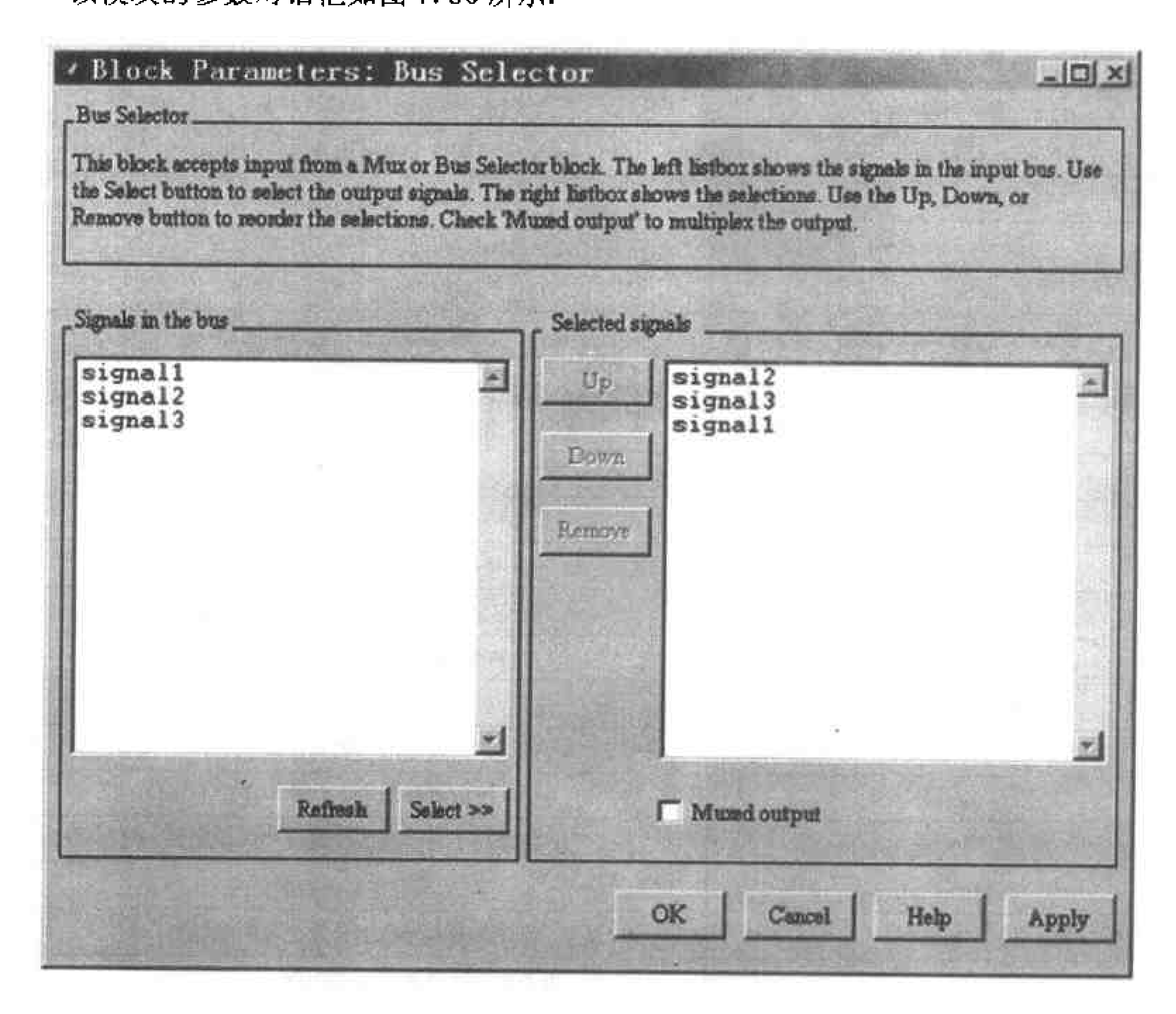


图 7.86　Bus Selector 模块参数对话框

1）总线上的信号(Signals in the bus)，总线上的信号列表框中显示输入总线的信号. 用 Select>>按钮选择从列表框中选择的输出信号.

2）选择的信号(Selected signals)，选择的信号列表框中显示输出信号，可以用 Up，Down 和 Remove 按钮对信号进行排序.

7.7.2　Configurable Subsystem(可配置子系统)

（1）模块功能

表达任何从用户指定的模块库中选择的模块.

（2）模块说明

Configurable Subsystem 模块可以表达任何包含于指定模块库中的模块. 其对话框允许指定其表达哪个模块，以及所表达模块的参数值.

Configurable Subsystem 模块简化了表达设计类模型的创建. 假设要建一个汽车模型，则提供发动机选择. 为这样的设计建模，首先创建一个可用于不同汽车的发动机的类型库，然后，在汽车模型中使用一个可配置子系统模块来表达发动机选择. 要建各种不同

的基本汽车模型，只要使用可配置发动机模块对话框，就可以选择发动机.

可配置子系统模块的外观随其代表的模块而变化. 初始情况下，可配置子系统模块不表示任何东西，此时，它没有端口. 如果选择了一个库和模块，可配置子系统显示图标和一系列与所选库中的输入和输出端口相应的输入和输出端口.

Simulink 映射库端口与可配置子系统模块端口的原则有：

1）库中每个惟一命名的输入/输出端口映射可配置子系统中独立的相同名字的输入/输出端口.

2）库中所有命名相同的输入/输出端口映射可配置子系统中同样的输入/输出端口.

3）用 Terminator 或 Ground 模块终止在当前选择的库模块中没有使用的任何输入/输出端口.

可配置子系统模块不提供与非输入/输出端口相应的端口，如触发和激活子系统中的触发和激活端口. 因此，不能使用可配置子系统模块来直接表达具有这些端口的模块. 但可以间接地包装该模块于一个子系统模块中，而该子系统可以具有非输入/输出端口.

（3）模块数据类型

该模块接受和输出信号类型，与其所表达的模块接受和输出的信号类型一样.

（4）模块参数对话框

Configurable Subsystem 模块参数对话框依据其当前是否表达了一个库及哪个模块而不同. 初始没有表达任何东西时，该模块的参数对话框如图 7.87 所示. 该对话框中仅显示一个空的库名称(Library name)参数.

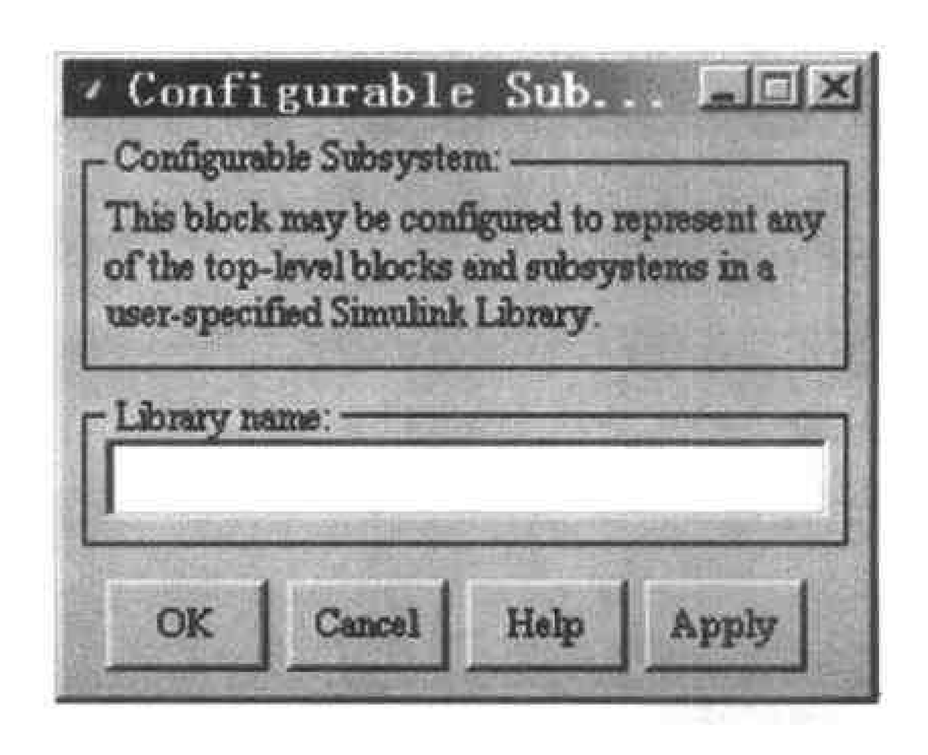


图 7.87　Configurable Subsystem 模块参数对话框

库名称(Library name)，可配置子系统表达的模块库的相对路径名. 如：simulink/Math.

不能使用模块对话框来改变已存在的配置子系统模块的库表达. 可以使用 set-param 模型创建命令，通过设置模块库参数来改变库为新库.

（5）模块特点

可配置子系统具有其所表达模块的特点.

7.7.3 Data Store Memory(数据存储器)

(1) 模块功能

定义一个数据存储器.

(2) 模块说明

Data Store Memory 模块定义并初始化一个命名共享数据存储器，它是一个对 Data Store Read 和 Data Store Write 模块有用的存储区域.

每一次数据存储都必须由 Data Store Memory 模块定义. 定义数据存储的 Data Store Memory 模块的位置决定了 Data Store Read 和 Data Store Write 模块能否访问数据存储.

如果 Data Store Memory 模块在顶层系统(top-level system)中，数据存储可以被模型中任何地方的 Data Store Read 和 Data Store Write 模块访问.

如果 Data Store Memory 模块在子系统(subsystem)中，数据存储只能被模型中同一子系统或在模型层次中比该子系统更低的子系统中的 Data Store Read 和 Data Store Write 模块访问.

可以通过指定 Initial value 参数的值来初始化数据存储. 值的长度决定了数据存储的长度. 如果 Data Store Write 模块没有写入相同长度的数据就会出现错误.

(3) 模块数据类型

该模块存储双精度类型的实数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.88 所示.

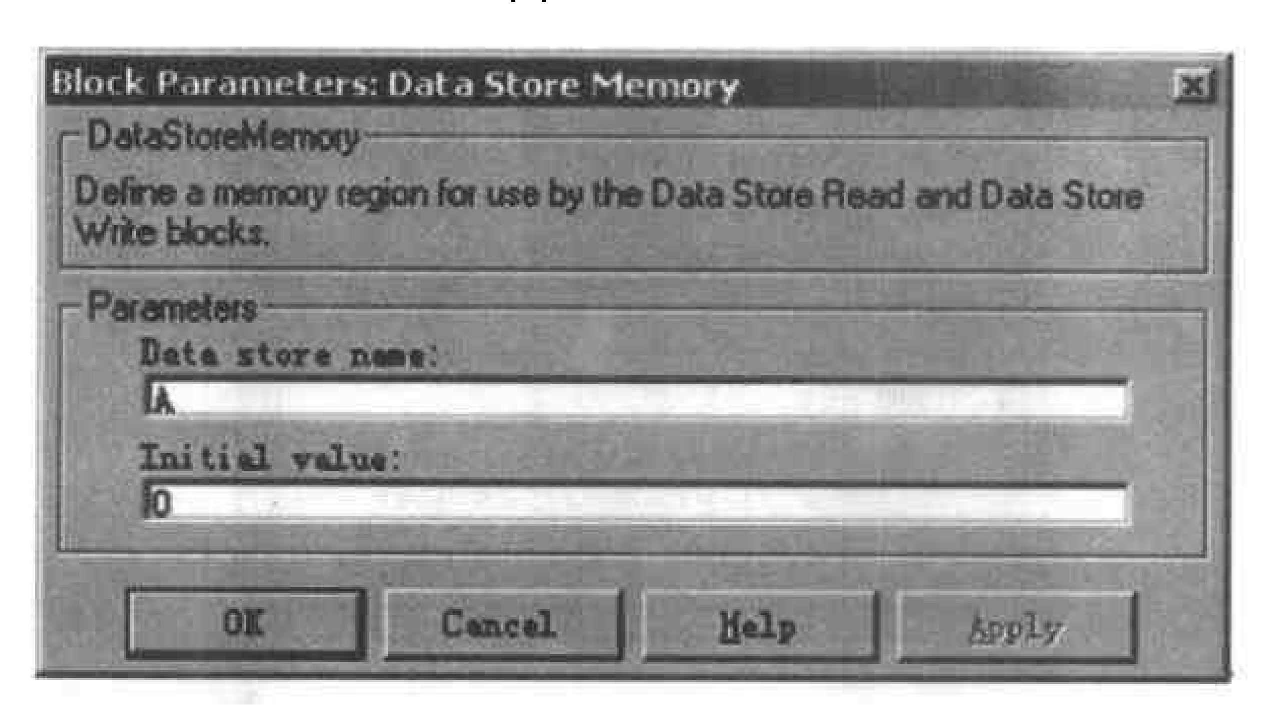


图 7.88 Data Store Memory 模块参数对话框

1) 数据存储名字(Data store name)，定义数据存储的名字. 缺省值是 A.

2) 初始值(Initial value)，数据存储初始值. 缺省值为 0.

(5) 模块特点

1) 采样时间 N/A；

2) 可向量化.

7.7.4 Data Store Read(读数据存储)

(1) 模块功能

从一个数据存储中读数据.

(2) 模块说明

Data Store Read 模块从一个被指名的数据存储中读取数据,并将数据传送给输出.数据存储中的数据,事先由 Data Store Memory 模块初始化,并且由 Data Store Write 模块写入了数据.

要读数据的数据存储,由定义该数据存储的 Data Store Memory 模块的位置确定.

多个 Data Store Read 模块可以从同一个数据存储中读取数据.

(3) 模块数据类型

该模块输出双精度类型的实数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.89 所示.

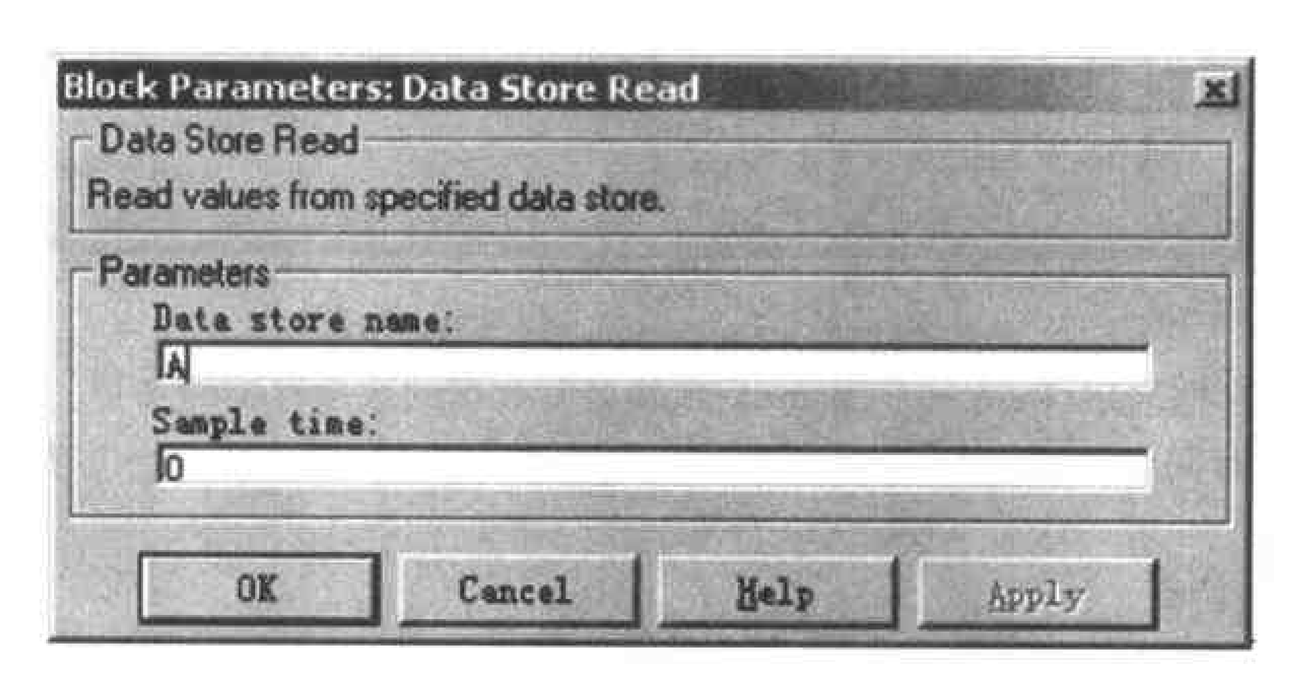


图 7.89 Data Store Read 模块参数对话框

1) 数据存储名字(Data store name),模块要读取数据的数据存储的名字.

2) 采样时间(Sample time),控制模块何时读数据存储.缺省值为 -1,表示采样时间是继承的.

(5) 模块特点

1) 采样时间连续或离散;

2) 可向量化.

7.7.5 Data Store Write(写数据存储)

(1) 模块功能

向一个数据存储写数据.

(2) 模块说明

Data Store Write 模块将其输入写入被指名的数据存储.

Data Store Write 模块对数据存储的每一次写入操作都会覆盖以前的内容.

要写入数据的数据存储由定义该数据存储的 Data Store Memory 模块的位置确定.数

据存储的大小由定义并初始化该数据存储的 Data Store Memory 模块设定. 每一个写入数据给数据存储的 Data Store Write 模块必须写入相同数量的数据.

多个 Data Store Write 模块可以向同一个数据存储中写入数据. 然而，如果在同一仿真步有两个 Data Store Write 模块试图给同一个数据存储写数据，结果将是不可预测的.

(3) 模块数据类型

该模块接受双精度类型的实数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.90 所示.

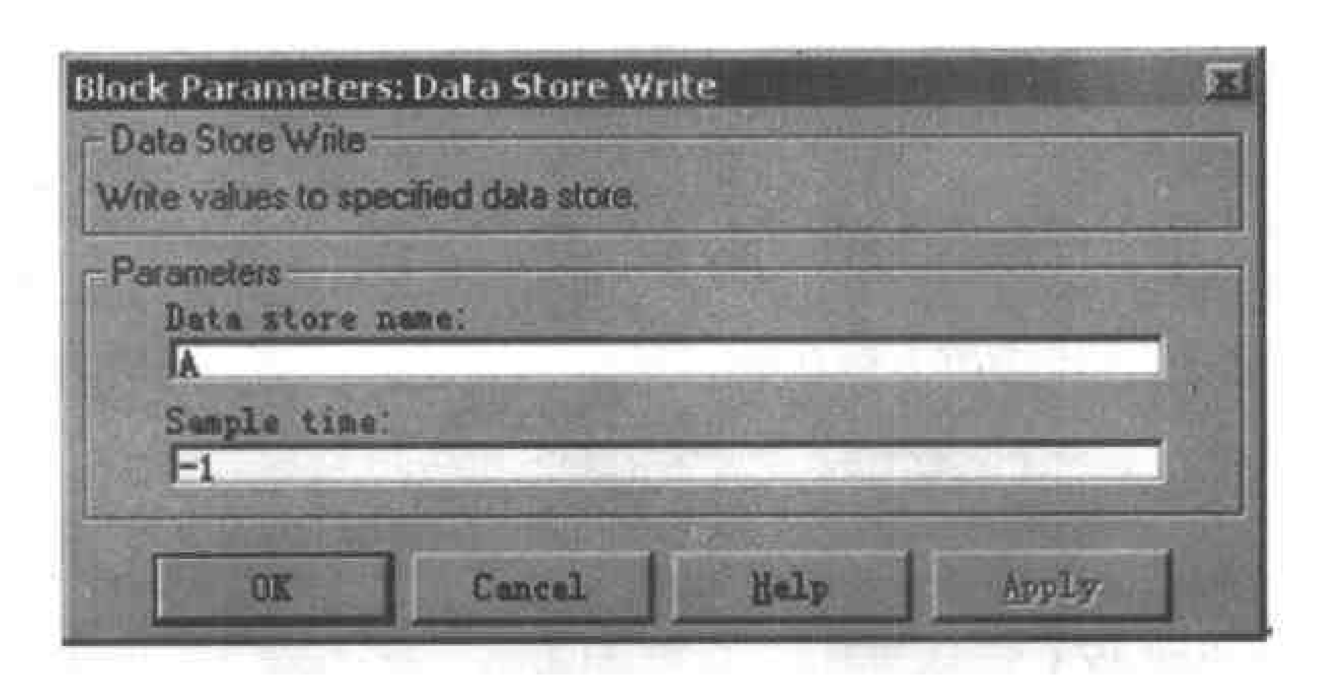


图 7.90 Data Store Write 模块参数对话框

1) 数据存储名字(Data store name)，模块要写数据的数据存储的名字.

2) 采样时间(Sample time)，控制模块何时写数据存储. 缺省值为－1，表示采样时间是继承的.

(5) 模块特点

1) 采样时间连续或离散；

2) 可向量化.

7.7.6 Data Type Conversion(数据类型转换)

(1) 模块功能

将输入信号转换为指定的数据类型.

(2) 模块说明

Data Type Conversion 模块将其输入信号转换为由 Data type 参数指定的数据类型.

(3) 模块数据类型

该模块输入可以是任何类型的实数或复数值信号. 如果输入是实数，输出是实数；如果输入是复数，输出也是复数.

(4) 模块参数对话框

该模块的参数对话框如图 7.91 所示.

1) 数据类型(Data type)，指定输入信号转换为何种类型. 可以选择的类型有：auto，double，single，int8，uint8，int16，uint16，int32，uint32，boolean. 选择 auto，将其输入信号转换为与其输出端口相连模块输入端口所需的类型.

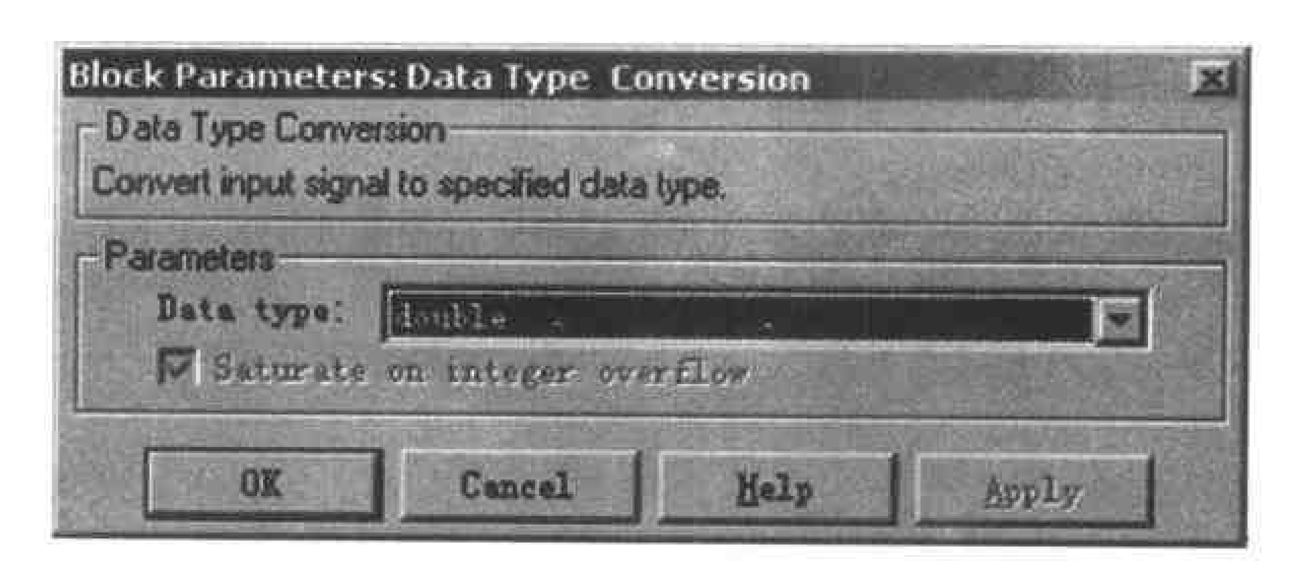


图 7.91 Data Type Conversion 模块参数对话框

2）整数饱和溢出(Saturate on integer overflow)，只有在整数输出时，才激活. 如果选择该核选框，如果整数溢出，将会输出饱和值.

(5) 模块特点

1) 直接馈通；

2) 采样时间由驱动模块继承；

3) 参数可以标量扩展；

4) 可向量化；

5) 有过零区间，检测何时到达极限.

7.7.7 Demux(解混)

(1) 模块功能

将一个向量信号分解成输出信号.

(2) 模块说明

Demux 模块将输入向量信号分解为若干输出连线，输出连线可以传输标量或者向量信号. Simulink 通过输出个数(Number of outputs)参数确定输出信号的数目和各个输出信号的宽度.

1) 输出标量数. 如果 Number of outputs 为标量，模块将输入信号分解为由该参数指定的若干个输出信号. 各个输出信号的宽度取决于输入向量的宽度和输出信号的个数：

如果输入信号的宽度等于输出的个数，该模块将输入信号向量分解成标量信号.

如果输入信号的宽度能够被输出的个数整除，该模块将输入信号分解成等宽度的向量信号.

如果输入信号的宽度不能被输出的个数整除，该模块将输入信号分解成宽度不等的向量信号，并且 Simulink 发出一条警告消息.

2) 输出的向量数. 如果 Number of outputs 参数为向量，输出线的条数等于 Number of outputs 向量的元素的个数. 输出信号的宽度取决于输入向量的宽度和 Number of outputs 参数中各个元素的值. 可以明确地指定输出信号的宽度或者由 Simulink 确定它们的宽度.

如果 Number of outputs 向量的各个元素都是正值，该模块生成的信号宽度为指定的宽度.

如果 Number of outputs 向量的各个元素中包含有正数和 −1 的值，该模块生成的输出信号中，那些宽度被指定为正数值的向量的宽度即为该数，而那些宽度被指定为负数的向量的宽度由 Simulink 动态地确定.

如果 Number of outputs 向量的所有元素都是 −1，输出的个数等于该参数向量的元素的个数，并且各个输出的宽度动态地被确定. 用这种方式指定参数与将参数指定为标量且它的值与参数向量的元素个数相等是一样的. 例如，指定参数值为[−1 −1 −1]与指定参数值为 3 是一样的.

Simulink 在该模块上画出指定个数的输出端口. 当端口的数目增加或减少时，从模块图标的下部增加或减去端口.

3) 指定输出参数由变量提供. 如果指定 Number of outputs 参数为一变量，且变量在工作空间中未被定义时，Simulink 将会发出一条错误消息.

(3) 模块数据类型

该模块接受和输出任何类型的实数或复数值信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.92 所示.

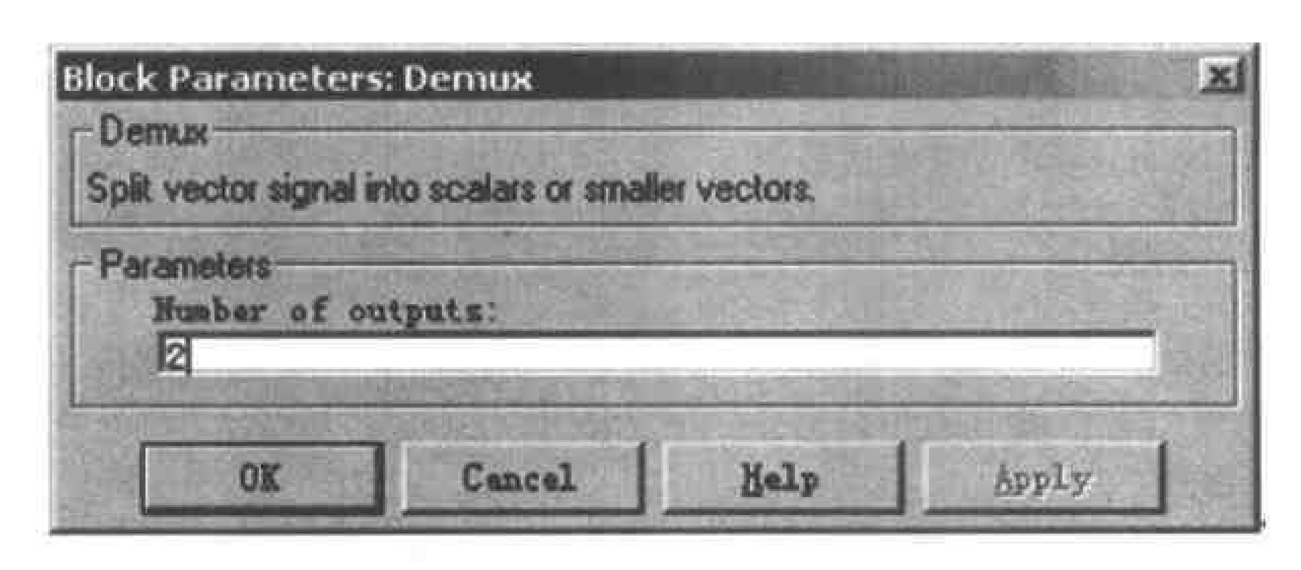


图 7.92　Demux 模块参数对话框

输出个数(Number of outputs)，指输出的个数和宽度. 所有输出宽度必须与输入线的宽度相匹配.

7.7.8　Enable(激活)

(1) 模块功能

给子系统加一个激活端口.

(2) 模块说明

给子系统增加 Enable 模块使之成为激活子系统. 激活子系统只有当它的激活输入端口的信号大于 0 时才运行.

在仿真开始时，Simulink 初始化激活子系统内的各个模块的状态为它们的初始状态. 当激活子系统重新启动时(在已经禁止后重新执行)，参数 States when enabling(激活时状态)可以确定包含在激活子系统内的各个模块的状态：

reset，将状态复位为它们的初始状态(如果没有定义则为 0).

held，保持状态为它们先前的值.

可以通过选择 Show output port(显示输出端口)核选框输出激活信号. 选择该选项允许系统处理激活信号. 激活信号的宽度即为该信号的宽度.

一个子系统最多只能包含一个 Enable 模块.

(3) 模块数据类型

该模块的输入可以是双精度或逻辑类型.

(4) 模块参数对话框

该模块的参数对话框如图 7.93 所示.

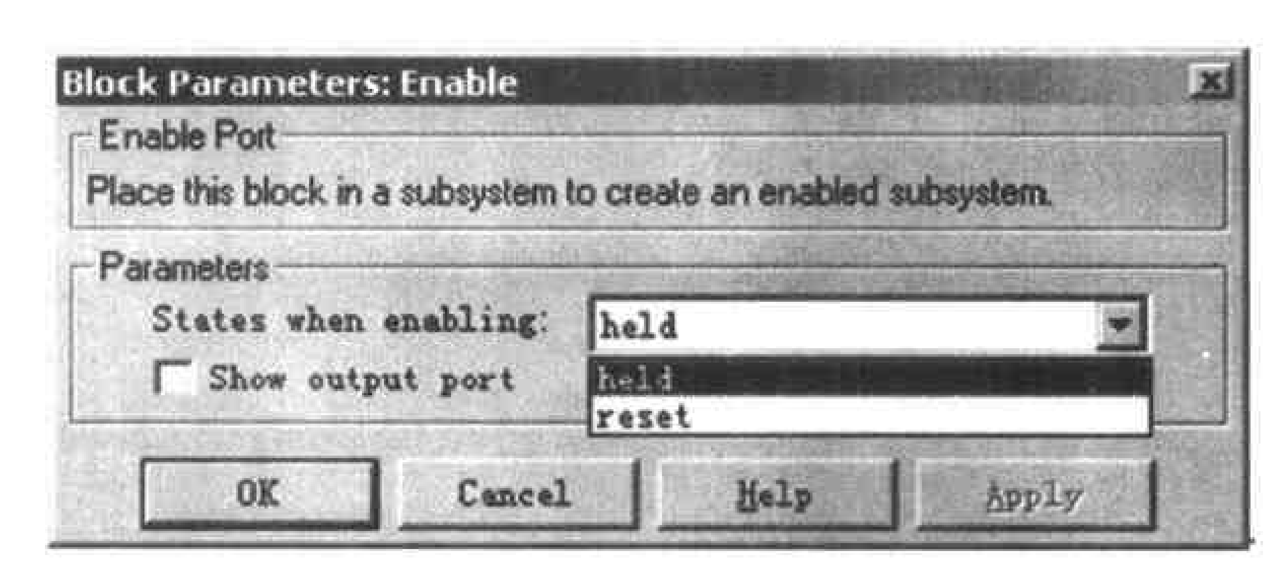


图 7.93　Enable 模块参数对话框

1) 激活时状态(States when enabling),指定当子系统重新激活时,如何处理内部状态.

2) 显示输出端口(Show output port),如果选择该核选框,Simulink 将在该模块上画一个输出端口.

(5) 模块特点

1) 采样时间由激活端口信号确定;

2) 可向量化.

7.7.9　From(导入)

(1) 模块功能

接收从 Goto 模块的输入.

(2) 模块说明

From 模块从相应的 Goto 模块接收信号,然后将它传出作为它的输出. 使用 From 和 Goto 模块可以从一个模块传送信号给另外一个模块而不用将它们直接连接起来. 每一个 From 模块都与一个 Goto 模块相关联. Goto 模块的输入传给 From 模块,From 模块又传给与它相连的模块. 要将一个 Goto 模块与一个 From 模块相关联,在 Goto tag 参数域中输入 Goto 模块的标记符即可.

一个 From 模块只能从一个 Goto 模块接收信号,而一个 Goto 模块能够传送信号给多个 From 模块.

相关联的 Goto 模块和 From 模块可以出现在模型中的任何地方,除了一个例外:如果两个模块中的一个在条件执行的子系统中,另外一个模块必须在同一个子系统中或者在它的下一层子系统(但不是另一个条件执行的子系统)中. 然而,如果一个 Goto 模块与

一个状态端口相连，信号可以传送给另一个条件执行的子系统中的 From 模块.

Goto 模块标记符的可见性决定了能够接收它的信号的 From 模块. 模块的图标表明了 Goto 模块标记符的可见性：

局部标记符的名字括于方括号([])之中.

范围标记符的名字括于大括号({})之中.

全局标记符的名字显示时没有另外的字符.

(3) 模块数据类型

该模块输出任何类型的实数或复数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.94 所示.

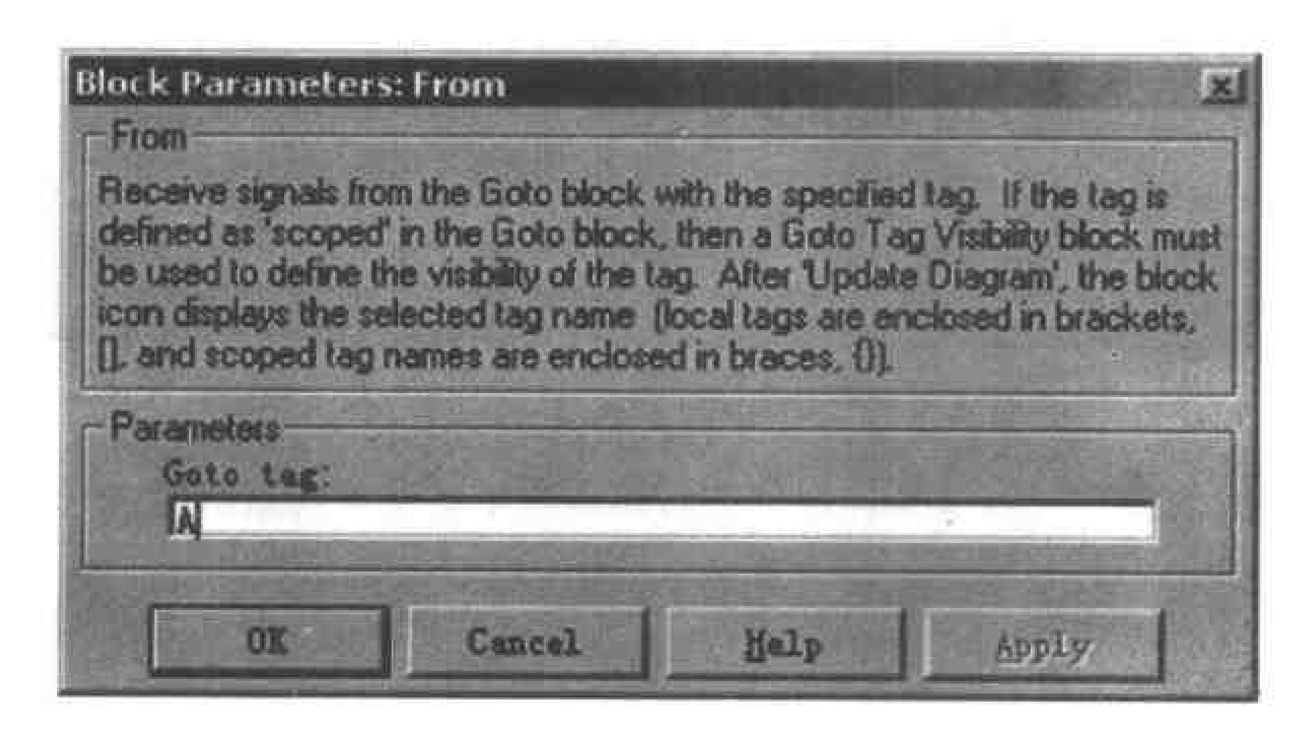


图 7.94 From 模块参数对话框

Goto tag 参数，传送信号给 From 模块的 Goto 模块的标记.

(5) 模块特点

1) 采样时间由驱动 Goto 模块的模块继承；

2) 可向量化.

7.7.10 Goto(传出)

(1) 模块功能

传送模块的输入给 From 模块.

(2) 模块说明

Goto 模块传送它的输入给与它相关联的 From 模块. 使用 From 和 Goto 模块可以将信号从一个模块传给另外一个模块而不用实际上直接将它们相连.

尽管一个 From 模块只能从一个 Goto 模块那儿接收信号，但是一个 Goto 模块可以将它的输入信号传给多个 From 模块. Goto 模块将它的输入传给一个与它相关联的 From 模块就像它们物理上连接在一起一样. 对于使用 From 和 Goto 模块的限制，在 From 模块中已介绍. Goto 模块和 From 模块通过使用 Goto 标记符(定义为 Tag 参数)相匹配.

Tag visibility 参数决定了获取信号的 From 模块的位置是否受到了限制：

local(缺省值),意味着使用标记符的 From 和 Goto 模块必须在同一个子系统中,局部标记符的名字被用方括号([])括了起来.

scoped 意味着使用标记符的 From 和 Goto 模块必须在同一个子系统中,或者在模型层次中任何低于 Goto Tag Visibility 模块的子系统.范围标记符的名字被用大括号({})括了起来.

global 意味着使用相同标记符的 From 和 Goto 模块可以在模型中的任何地方,全局标记符的名字显示时没有另外的字符.

如果使用相同标记符名字的 Goto 和 From 模块在同一个子系统中时使用局部标记符.如果使用相同标记符名字的 Goto 和 From 模块在不同的子系统中时必须使用全局标记符.如果定义一个标记符为全局的,所有使用这一标记符的模块获取同一信号.一个定义为范围的标记符能够在模型中的多个地方使用.

(3) 模块数据类型

该模块接受任何类型的实数或复数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.95 所示.

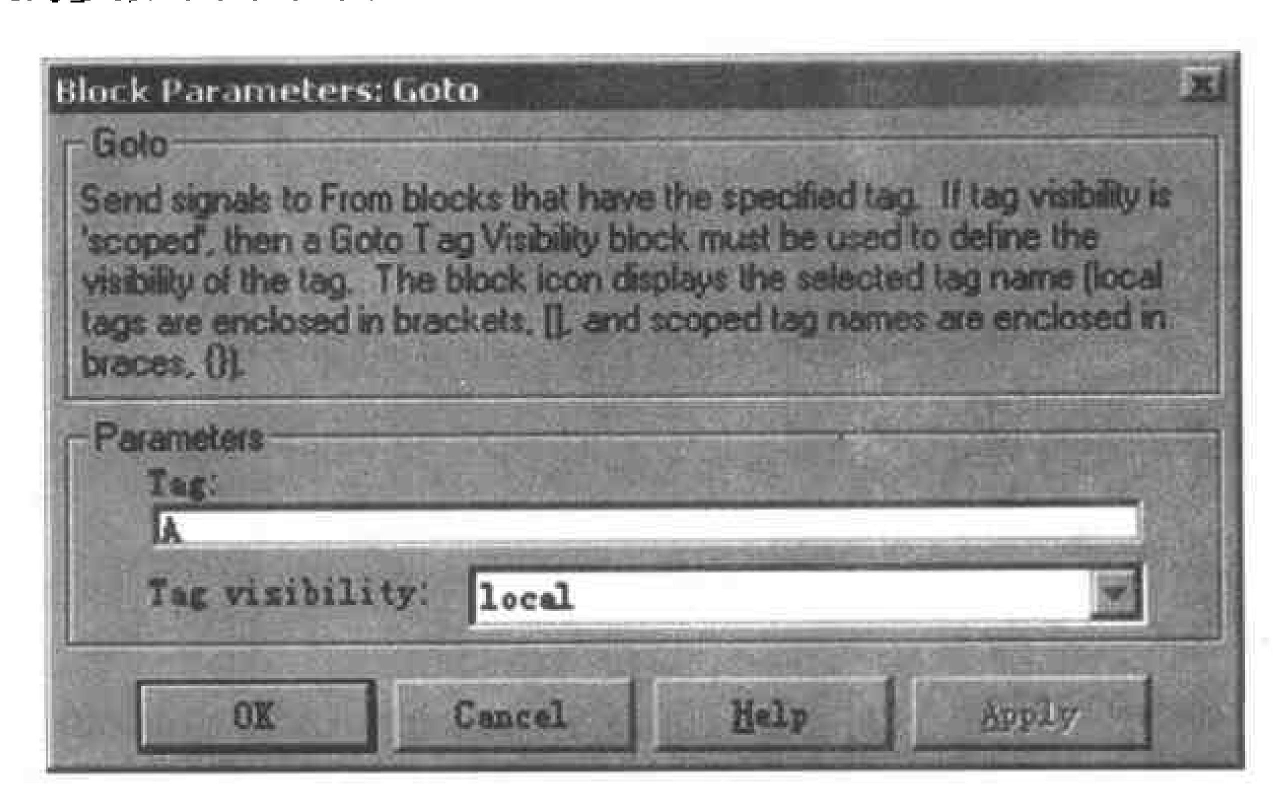


图 7.95　Goto 模块参数对话框

1) Tag 参数,是 Goto 模块的标识符.

2) Tag visibility 参数,Goto 模块标记的范围:local,scoped 或 global.缺省为 local.

(5) 模块特点

1) 采样时间由驱动模块继承;

2) 可向量化.

7.7.11　Goto Tag Visibility(传出标记符的可见性)

(1) 模块功能

定义 Goto 模块标记符的范围.

(2) 模块说明

Goto Tag Visibility 模块定义具有 scoped(范围)可见性的 Goto 模块标记符的可访问性. 指定为 Goto tag 参数的标记符能够被与 Goto Tag Visibility 模块包含在同一个子系统中的 From 模块或者在模型层次中比包含有 Goto Tag Visibility 模块的子系统层次更低的子系统中的 From 模块获取.

对于参数 Tag visibility 的值是 scoped 的 Goto 模块来说，Goto Tag Visibility 模块是必须的. 当标记符可见性是 local 或 global 时不需要该模块. 该模块的图标显示了用大括号({})括起来的标记符名字.

(3) 模块参数对话框

该模块的参数对话框如图 7.96 所示.

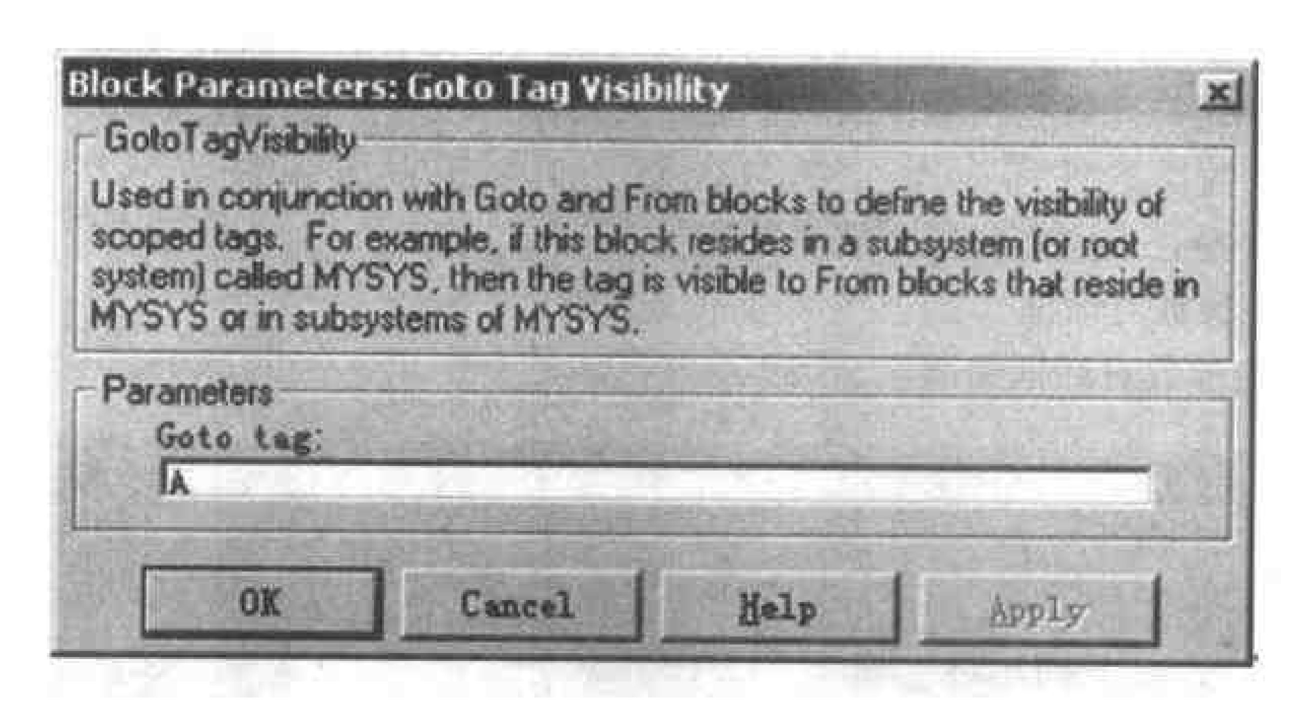


图 7.96　Goto Tag Visibility 模块参数对话框

Goto Tag 参数,Goto 模块的标记.

7.7.12　Ground(接地)

(1) 模块功能

给悬空的输入端口接地.

(2) 模块说明

Ground 模块可以用来连接那些输入端口没有与其它的模块相连的模块. 如果有模块的输入端口悬空时运行仿真,Simulink 将会发出警告信息. 使用 Ground 模块将那些模块接地以避免警告信息. Ground 模块输出的信号值为 0.

(3) 模块数据类型

该模块输出与其相连端口的数据和数值类型一致的信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.97 所示.

(5) 模块特点

1) 采样时间由驱动模块继承;

2) 可向量化.

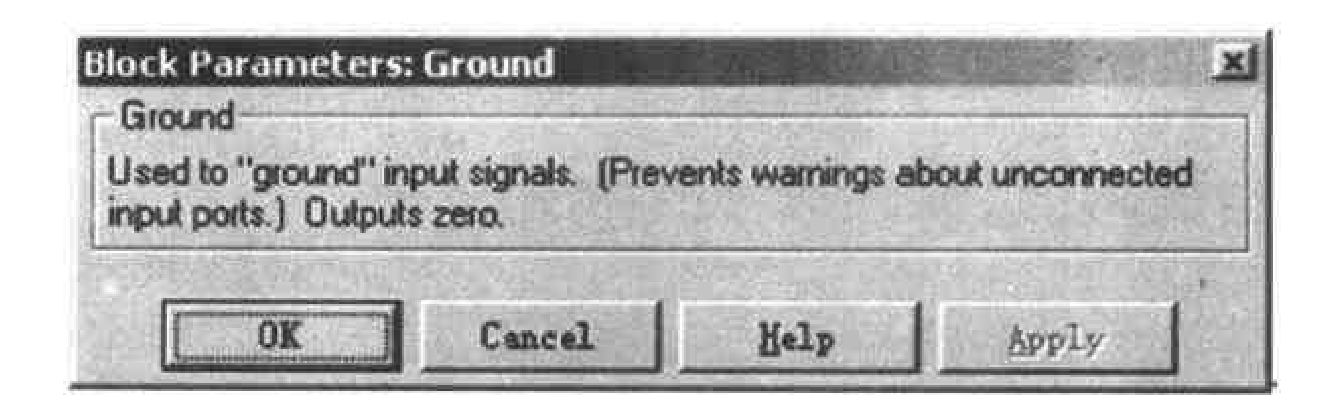


图 7.97　Ground 模块的对话框

7.7.13　Hit Crossing(捕获穿越点)

(1) 模块功能

检测穿越点.

(2) 模块说明

Hit Crossing 模块检测输入是何时按照 Hit crossing direction 参数给出的方向到达了由 Hit crossing offset 参数给定的值. 该模块寻找在机器误差容限范围内的穿越点.

该模块有一个输入. 如果选取了 Show output port 核选框,模块的输出表明穿越是什么时候发生的. 如果输入信号在某一时间步的值刚好等于偏移值,模块在那一时间步的输出值为 1. 如果输入信号的某相邻两点将偏移值包括在中间(但是都不正好等于偏移量),模块在第二个时间步输出 1. 如果没有选取 Show output port 核选框,模块确保仿真找到穿越点但并不产生输出.

Hit Crossing 模块的作用就像一个"几乎等于"模块,在数学和计算机精度有限时计算圆整容限比较有用,这一模块比在模型中加入逻辑以检测这一状态更为方便.

在 hardstop 和 clutch 演示例子中说明了如何使用 Hit Crossing 模块. 在 hardstop 示例中,Hit Crossing 模块包含在 Friction Model 子系统中. 在 clutch 示例中,Hit Crossing 模块在 Lockup Detection 子系统中.

(3) 模块数据类型

该模块输出逻辑类型信号,在逻辑兼容模式下,模块输出双精度信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.98 所示.

1) 穿越偏移量(Hit crossing offset)参数,检测穿越的值.

2) 穿越方向(Hit crossing direction)参数,输入信号接近穿越值的方向.

3) 显示输出端口(Show output port)参数,如果选中,画一个输出端口.

(5) 模块特点

1) 直接馈通;

2) 采样时间由驱动模块继承;

3) 可以标量扩展;

4) 可向量化;

5) 有过零区间,检测穿越.

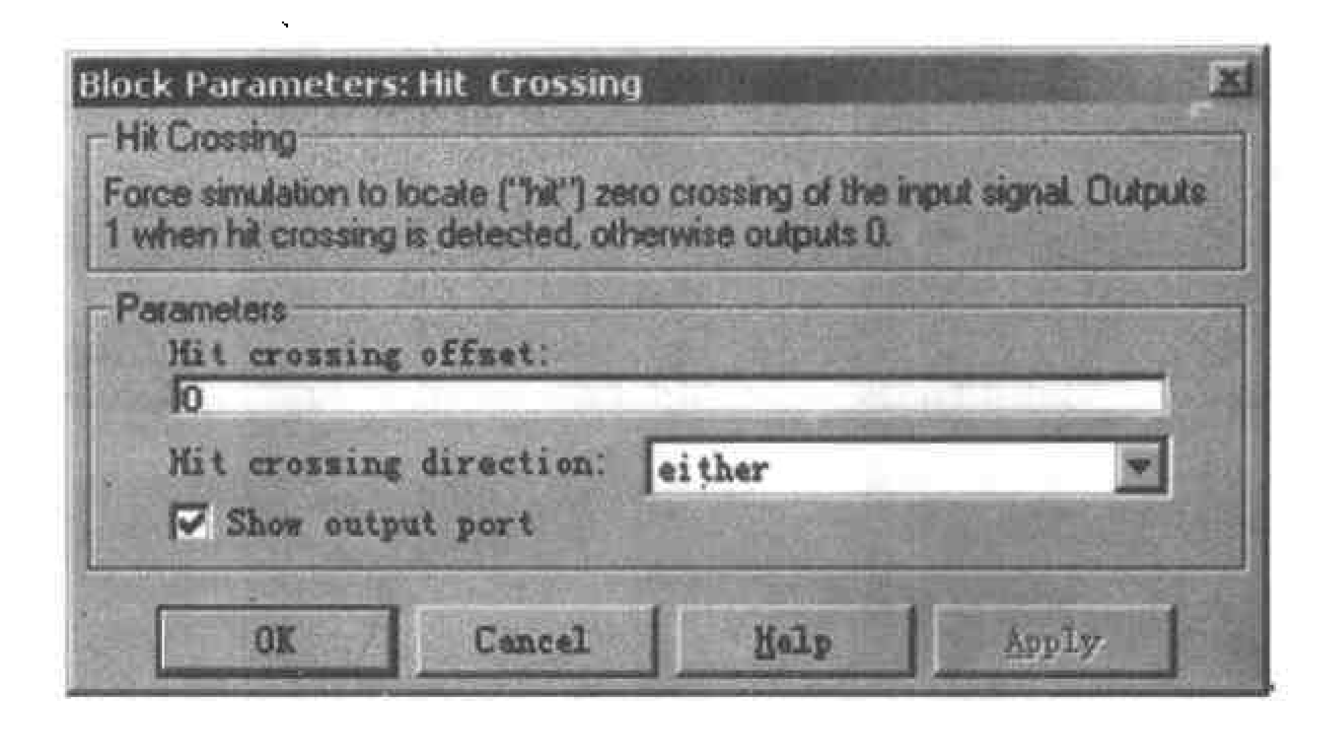


图 7.98 Hit Crossing 模块参数对话框

7.7.14 IC(初始状态)

(1) 模块功能

设置一个信号的初始值.

(2) 模块说明

IC 模块设置与它的输出端口相连的信号的初始状态.

IC 模块在为循环中的代数状态变量提供初始估计时也是有用的.

(3) 模块数据类型

该模块接受和输出双精度类型的信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.99 所示.

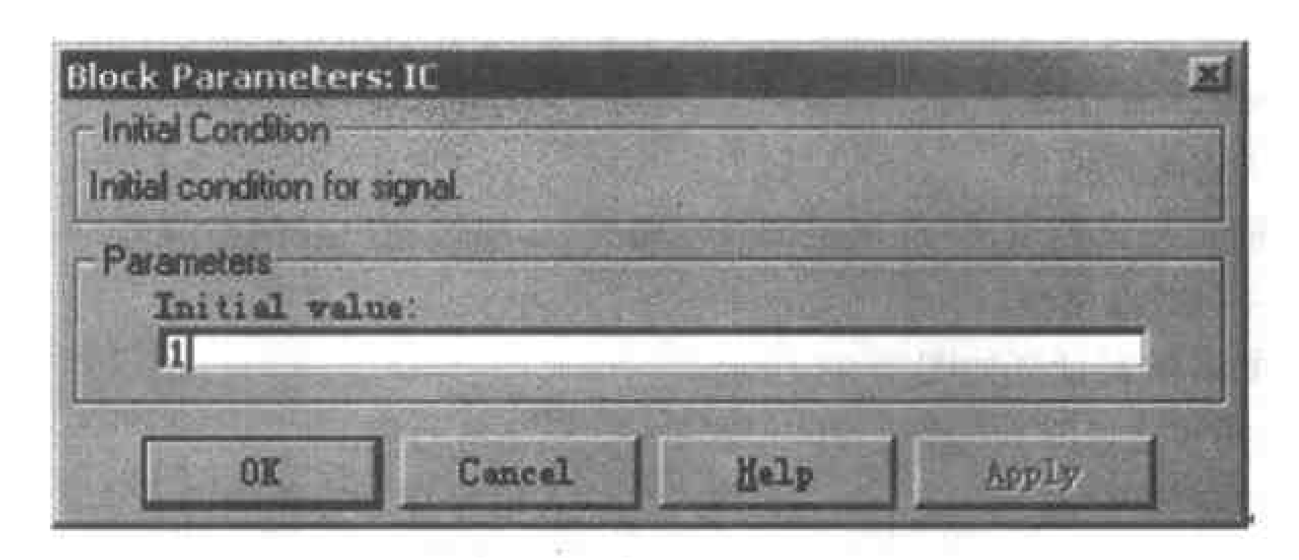


图 7.99 IC 模块参数对话框

初始值(Initial value),信号的初值.缺省值为 1.

(5) 模块特点

1) 直接馈通;

2) 采样时间由驱动模块继承;

3) 参数可以标量扩展;

4）状态 0：

5）可向量化；

6）没有过零区间.

7.7.15 Inport（输入端口）

（1）模块功能

为子系统或外部输入创建输入端口.

（2）模块说明

输入是从一个系统的外部引入信号到系统的内部.

Simulink 根据下面这些规则指定 Inport 模块端口的号码：

在顶层系统或者子系统中自动地依次给 Inport 模块编号码，从 1 开始.

如果增加一个 Inport 模块，它赋给下一个可用的号码.

如果删除一个 Inport 模块，另外的端口将会自动地重新排号以确保 Inport 模块按顺序排号并且中间不会有遗漏的号码.

如果拷贝一个 Inport 模块到系统，它的端口号码不会被重新编排，除非它的号码与系统中已有的 Inport 模块相冲突. 如果拷贝的 Inport 模块的端口号码不是按顺序的，必须重新给模块编号否则在运行仿真或更新模块图时会得到一个出错消息.

如果 Inport 模块提供一个向量信号，可以通过 Port width 参数指定 Inport 模块的输入宽度或者将 Port width 设为 -1（缺省值），让 Simulink 自动地确定输入宽度.

Sample time 参数是信号进入系统的速率. 缺省值为 -1，使得模块从驱动它的模块那儿继承采样时间. 在一个顶层系统中或在某些模型中，它的驱动 Inport 模块的采样时间没有确定时，最好为 Inport 模块设置 Sample time 参数.

1）子系统中的 Inport 模块. 子系统中的 Inport 模块代表子系统的输入. 信号到达 Subsystem 模块的一个端口时从那个子系统中与之相关的 Inport 模块流出. 与 Subsystem 模块的输入端口相关联的 Inport 模块的 Port number 参数与该 Subsystem 模块的输入端口的相对位置相一致. 例如，Port number 参数为 1 的 Inport 模块从与它所在的 Subsystem 模块的最上面的一个端口相连的模块那儿得到信号.

如果给 Inport 模块的 Port number 重新编号，该模块将与另外一个输入端口相连，尽管该模块继续接收子系统外的同一个模块的信号.

Inport 模块的名字显示在 Subsystem 模块的图标中作为端口的标签. 要禁止显示标签，选取 Inport 模块并从 Format 菜单中选择 Hide Name 菜单项，然后从 Edit 菜单中选择 Update Diagram 菜单项.

2）在顶层系统中的 Inport 模块. 顶层系统中的 Inport 模块有两个用处：从工作空间中提供外部输入，可以使用 Simulation Parameters 对话框或者 sim 命令做到这一点；提供扰动模型的方法：

要提供从工作空间的外部输入，使用 Simulation Parameters 对话框. 或者 sim 命令的 ut 参数.

如果指定时间和输入值的矩阵，第一列应该是时间值的向量，剩下的列是在那些时间点处的数据，每一列按照对应的端口号码为一个 Inport 模块提供数据. 如果有必要，在时

间向量中没有的时间点处，Simulink 使用插值得到输入数据.

要通过 linmod 和 trim 分析函数提供一个对模型进行扰动的方法. Inport 模块定义在哪些地方将输入引入模块.

（3）模块数据类型

该模块接受任何类型的实数或复数值信号.

（4）模块参数对话框

该模块的参数对话框如图 7.100 所示.

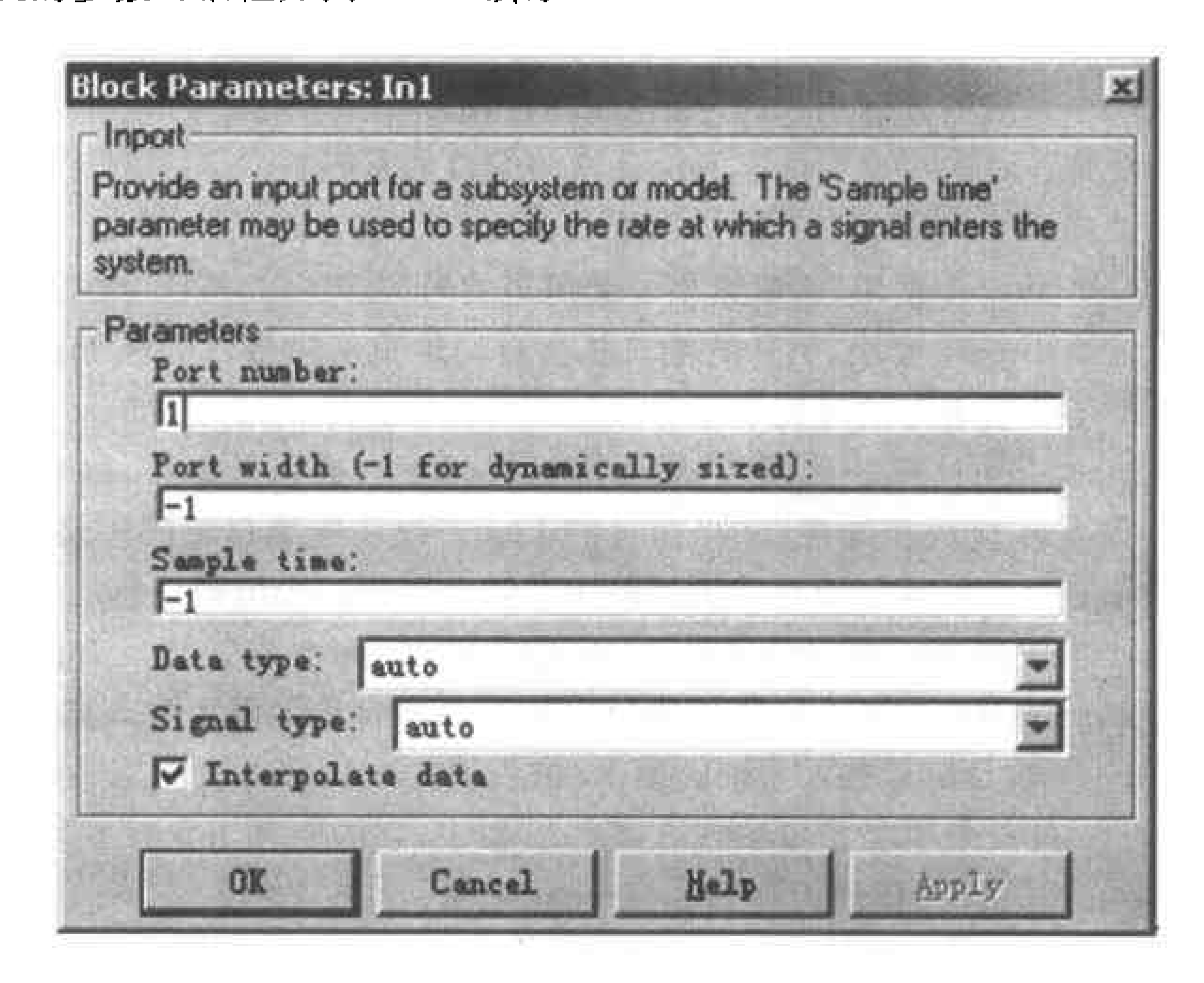


图 7.100 Inport 模块参数对话框

1）端口数(Port number)，指输入模块的端口数.

2）端口宽度(Port width)，输入端口输入信号的的宽度. 指定为 −1，自动确定.

3）采样时间(Sample time)，信号进入系统的速率.

4）数据类型(Data type)，指外部输入数据类型. 仅用于顶层输入端口，在子系统的输入端口对话框中不出现.

5）信号类型(Signal type)，指外部输入信号的(实数或复数)类型. 仅用于顶层输入端口，在子系统的输入端口对话框中不出现.

6）插值数据(Interpolate data)，选择插值或外推方法. 仅用于顶层输入端口，在子系统的输入端口对话框中不出现.

（5）模块特点

1）采样时间由驱动模块继承；

2）可向量化.

7.7.16 Merge(合并)

（1）模块功能

将输入连线组合为标量输出连线.

(2) 模块说明

Merge 模块将其输入连线合并为单个输出线，其任何时刻的输出值与其驱动模块的最近计算输出相等. 通过指定输入个数(Number of Inputs)参数，来指定该模块输入的个数. 所有输入必须宽度一样，如所有标量或所有向量宽度都是 4. 可以通过设置初始输出参数来指定模块的初始输出. 如果不指定初始输出，而其一个或多个驱动模块指定了初始输出，Merge 模块的初始输出将与驱动模块初始输出的最近的估计值相等. Merge 模块为交替执行的子系统创建提供了方便.

Simulink 限制了与 Merge 模块输入的连接，从非虚拟模块输出到 Merge 模块的输入，只允许建立一对一映射的连接. 如，可以使用 Goto/From 模块对，连接模型图一部分中的非虚拟模块的标量和向量输出和另一部分中的 Merge 模块的输入. 不能使用 Merge 模块连接多个非虚拟输出. Simulink 在仿真开始时会检测模块图中连接的合法性. 如果检测到不合法的连接，会停止仿真，并显示错误信息.

(3) 模块数据类型

Merge 模块接受任何数值(复数或实数)和数据类型的信号，包括用户自定义类型. 如果输入类型为用户自定义，初始条件必须为 0.

(4) 模块参数对话框

Merge 模块的参数对话框如图 7.101 所示.

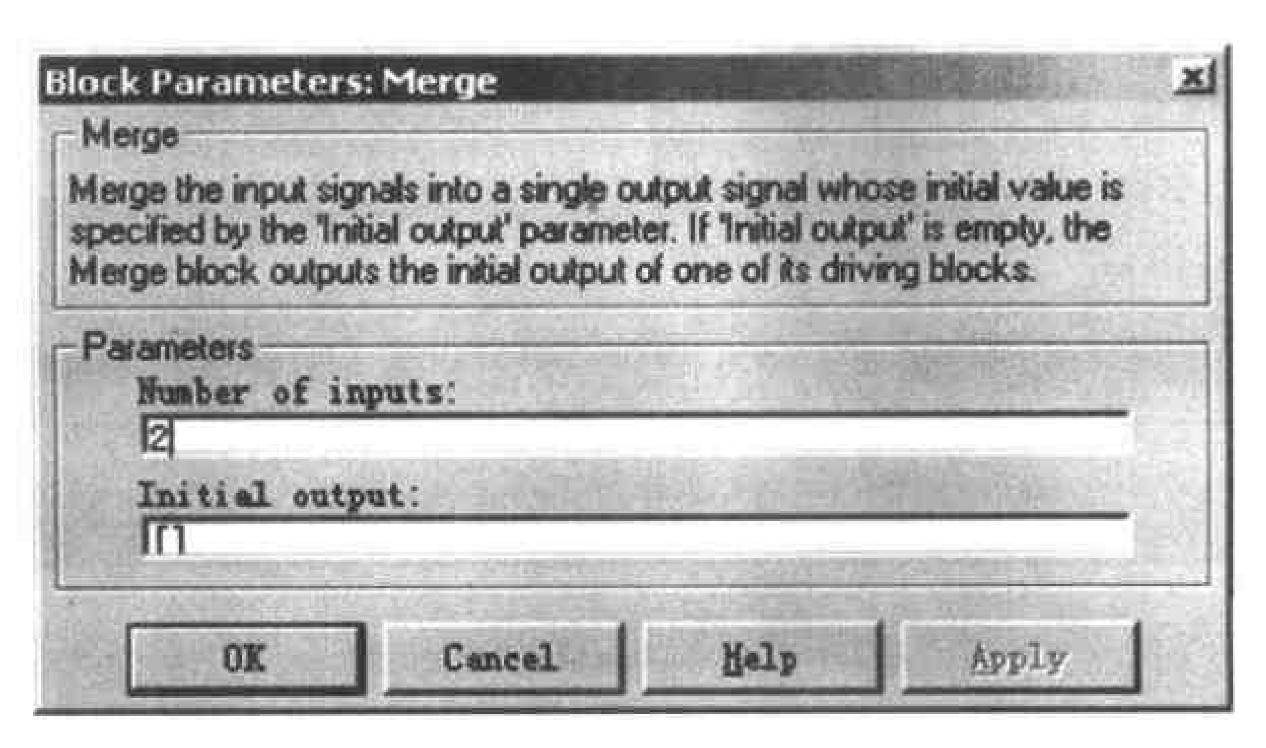


图 7.101 Merge 模块参数对话框

1) 输入个数(Number of inputs)，要合并的输入端口的个数. 端口可以是标量或向量.

2) 初始输出(Initial output)，Merge 模块的初始输出值. 如果没有指定，与驱动模块的初始输出相等.

(5) 模块特点

1) 采样时间由驱动模块继承；

2) 可向量化；

3) 没有标量扩展.

7. 7. 17 Model Info(模型信息)

(1) 模块功能

显示模型修改控制信息.

(2) 模块说明

Model Info 模块将在模型的模块图中显示修改控制信息,作为模块的注释.

(3)模块参数对话框

该模块的参数对话框如图 7. 102 所示.

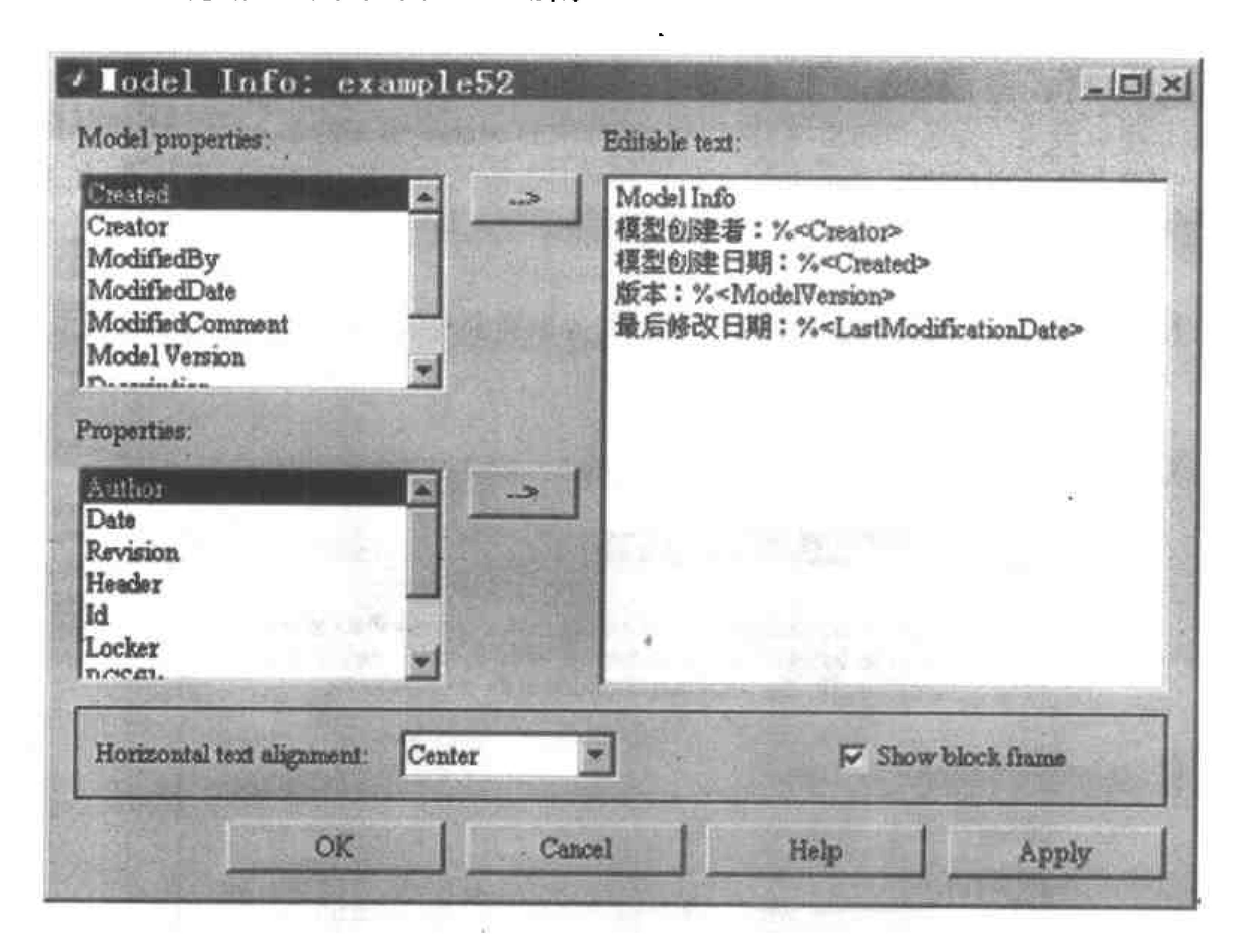


图 7. 102 Model Info 模块参数对话框

1) 可编辑文本(Editable text),输入在 Model Info 模块中显示的文本,可以用%<变量名>的形式自由嵌入变量名. 变量名可以是模型名字,修改控制信息. 变量值会在显示文本中显示.

2) 模型属性(Model properties),是模型中保存的修改控制属性列表. 从列表中选择一个属性,并用邻近的箭头按钮将该属性对应的变量输入到可编辑文本中去.

3) 属性(Properties),只有在模型属性中指定了外部配置管理者后,该域才能出现.

7. 7. 18 Mux(混合)

(1) 模块功能

将几条输入信号线组合成一条向量信号线.

(2) 模块说明

Mux 模块将几条输入信号线组合成一条向量信号线. 每一输入信号线能够承载一个标量信号或者一个向量信号. Mux 模块的输出是向量.

如果定义 Number of inputs 参数为标量,Simulink 通过检查 Mux 模块的驱动模块的输出端口以确定 Mux 模块的输入宽度. 如果任何一个输入是向量,它的所有元素都由该模块组合在一起.

如果有必要明确地定义输入的宽度,可以指定他们为向量. 对于那些宽度需要仿真时动态确定的输入将−1 作为它的元素. 如果输入信号的宽度与期望的不一致,Simulink 将会显示一条出错消息.

例如,[4 1 2]表明有三个输入形成了一个有 7 个元素的输出向量:前四个输出元素来自第一个输入,第五个元素来自第二个输入,第六、第七个元素来自第三个输入. 如果这些输入是否具有固定宽度并不是很重要,可以指定 Number of inputs 为 3.

要指定三个输入,其中第一个输入向量必须有四个元素,可以指定[4 −1 −1]. Simulink 根据第二、第三个输入的宽度确定输出的宽度.

Simulink 在 Mux 模块的图标上画上指定数目的输入. 如果改变了输入端口的数目,Simulink 在模块图标的下部加上或者减去端口.

在使用变量提供 Number of Inputs 参数时,即指定 Number of inputs 参数为变量时,如果变量在工作空间中未被定义,Simulink 就会发出出错消息.

(3) 模块数据类型

该模块接受任何类型的复数或复数信号,包括混合类型向量.

(4) 模块参数对话框

该模块的参数对话框如图 7.103 所示.

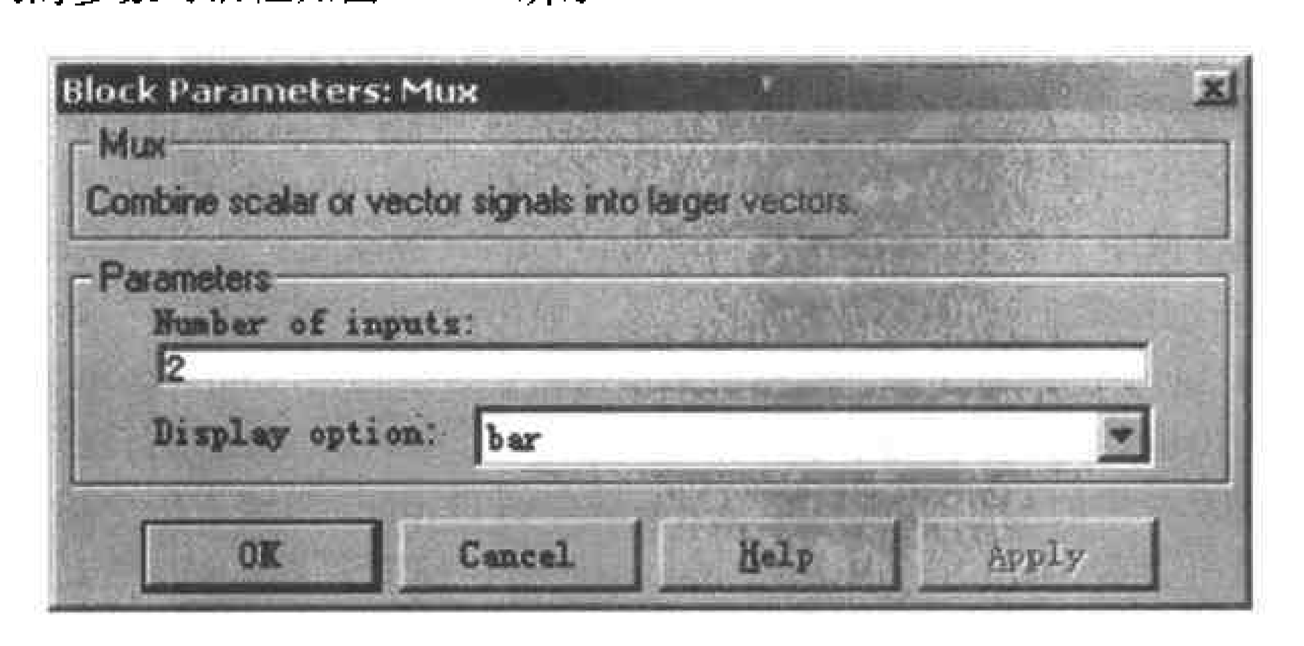


图 7.103 Mux 模块参数对话框

1) 输入个数(Number of inputs),输入的个数和宽度. 输出线的宽度等于输入线宽度之和. 当显示选项参数为 names 时,可以在该参数域中输入逗号分开的信号名列表.

2) 显示选项(Display option),在模型中显示模块的图标. 其选项有三:

none: Mux 出现在模块图标中;

names: 在每个端口附近显示信号名;

bar: 以实心前景颜色显示图标,缺省选项.

7.7.19 Outport(输出端口)

(1) 模块功能

为子系统或者外部输出创建一个输出端口.

(2) 模块说明

输出端口是从系统到系统外的目标的连接. Simulink 根据如下的规则分配 Outport 模块的编号:

在顶层系统或者子系统中自动地依次给 Outport 模块编号,编号从 1 开始;

如果增加一个 Outport 模块,它赋给下一个可用的号码;

如果删除一个 Outport 模块,另外的端口将会自动地重新排号以确保 Inport 模块按顺序排号并且中间不会有遗漏的号码;

如果拷贝一个 Outport 模块到一个系统,它的端口号码不会被重新编号,除非它的号码与系统中已有的 Outport 模块相冲突. 如果拷贝的 Outport 模块的端口号码不是按顺序的,必须重新给模块编号否则在运行仿真或更新模块图时会出现出错消息.

1)子系统中的 Outport 模块. 子系统中的 Outport 模块代表子系统的输出. 到达子系统的 Outport 模块的信号从该 Subsystem 模块的相关输出端口传出. 与 Subsystem 模块的输出端口相关联的 Outport 模块的 Port number 参数与 Subsystem 模块的输出端口的相对位置一致. 例如,Port number 参数为 1 的 Outport 模块将它的信号送到与 Subsystem 模块最上端的输出端口相连的模块.

如果给 Outport 模块的 Port number 重新编号,该模块将与另外一个输出端口相连,尽管该模块继续发送信号到子系统外的同一个模块.

在通过选择已存在的一些模块创建子系统时,如果有多个 Outport 模块包含在各个分组模块中,Simulink 自动地给这些模块的各个端口重新编号.

Outport 模块的名字显示在 Subsystem 模块的图标中作为一个端口的标签. 要禁止显示标签,选取 Outport 模块并从 Format 菜单中选择 Hide Name 菜单项.

2)条件执行的子系统中的 Outport 模块. 当 Outport 模块在一个可触发或者激活子系统中时,当子系统被禁止时可以指定如何处理其输出:它可以被复位为初始值,也可以保持它最近的值. Output when disabled 弹出菜单提供了这些选项. Initial output 参数是子系统执行前的输出值,如果 reset 选项被选取时,它还是子系统被禁止时的输出值.

3)在顶层系统中的 Outport 模块. 顶层系统中的 Outport 模块有两个用处:向工作空间提供外部输出,可以使用 Simulation Parameters 对话框或者 sim 命令做到这一点;并为分析函数提供从系统中获取输出的方法.

提供到工作空间的外部输出(使用 Simulation Parameters 对话框). 在 Workspace I/O 页面中,在 Save to wordspace 域中选取 Output 核选框并且在数据域中指定变量.

如果在编辑框中指定一个变量名,所有的输入都写入该变量. 如果系统有多个 Outport 模块,变量将是一个矩阵,它的每一列包含一个不同 Outport 模块的数据. 列的顺序与 Outport 模块的端口编号的顺序一致. 如果指定多个变量名,不同的 Outport 模块产生的数据写给不同的变量. 例如,如果系统中有两个 Outport 模块,要将 Outport 模块 1 的数据保存在 speed 变量中,将 Outport 模块 2 的数据保存在 dist 变量中,指定 Output 参数为

speed,dist.

提供外部输出给工作空间(使用 sim 命令).可以使用第三个返回参数将输出写到工作空间:[t, x, y] = sim(...).

如果系统中有多个 Outport 模块,上面的命令得到的 y 是一个矩阵,它的不同的列包含不同的 Outport 模块产生的数据.列的顺序与 Outport 模块的端口编号的顺序一致.

如果在第二个(状态)参数后指定多个变量名,不同的 Outport 模块产生的数据写入不同的变量.例如,如果系统有两个 Outport 模块,要将 Outport 模块 1 的数据保存在 speed 变量中,将 Outport 模块 2 的数据保存在 dist 变量中,用如下的命令:

[t, x, speed, dist] = sim(...);

为 linmod 和 trim 分析函数提供从系统获取输出的方法.

(3) 模块数据类型

该模块接受任何 MATLAB 数据类型的实数或复数信号作为输入.

(4) 模块参数对话框

该模块的参数对话框如图 7.104 所示.

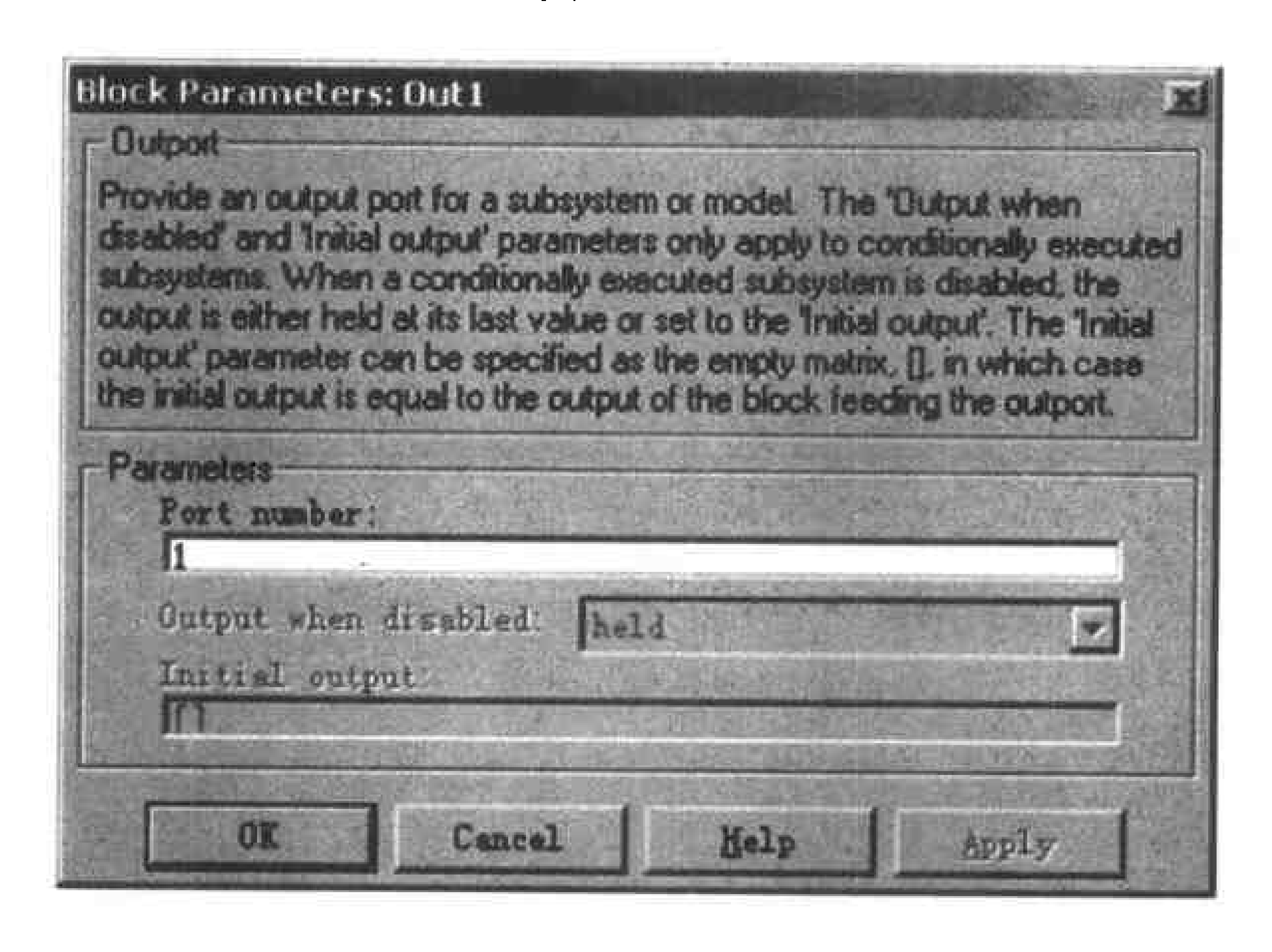


图 7.104　Output 模块参数对话框

1) 端口数(Port number),输出端口模块的端口数.缺省为 1.

2) 禁止时输出(Output when disabled),对于条件执行子系统,当系统禁止时的输出.

3) 初始输出(Initial output),对于条件执行子系统,子系统禁止后执行前模块的输出.

(5) 模块特点

1) 采样时间由驱动模块继承;

2) 可向量化.

7.7.20 Probe(探测器)

(1) 模块功能

检测一个信号连线的宽度、采样时间、复数信号标志.

(2) 模块说明

Probe 模块选择性地输出其输入信号的信息.输出包括:输入信号的宽度、采样时间、输入复数值标志.该模块只有一个输入端口,其输出端口数取决于所选择的探测信息.每个探测值就是在不同端口上的一个独立的输出信号.仿真时该模块显示探测数据.

(3) 模块数据类型

该模块接受和输出双精度类型的信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.105 所示.

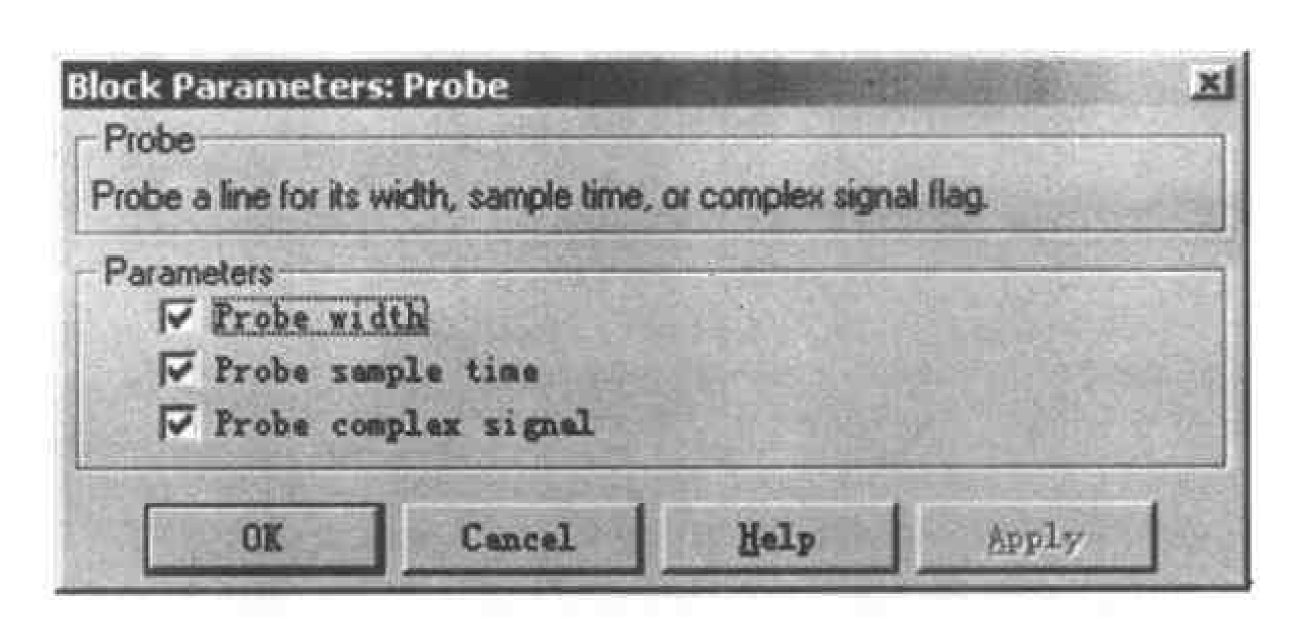


图 7.105 Probe 模块参数对话框

1) 探测宽度(Probe width),如果选择,输出探测连线的宽度.

2) 探测采样时间(Probe sample time),如果选择,输出探测连线的采样时间.

3) 探测复数信号(Probe complex signal),如果选择,探测信号是复数,输出 1,否则为 0.

(5) 模块特点

1) 直接馈通;

2) 采样时间由驱动模块继承;

3) 可以标量扩展;

4) 可向量化;

5) 没有过零区间.

7.7.21 Selector(选择器)

(1) 模块功能

选择输入元素.

(2) 模块说明

Selector 模块将选取的输入向量的元素作为它的输出.

Elements 参数定义输出向量中输入向量元素的顺序. 该参数必须指定为一向量,除非只选取了一个元素. 如图 7.106 所示的模型,显示了 Selector 模块的图标和它在输入向量为[1 3 5 7 9 11]并且 Elements 参数值为[6 2 4 1]时的输出.

模块的图标以图形化的形式显示输入向量被选取元素的顺序. 如果模块的图标不够大,就显示模块的名字.

(3) 模块数据类型

该模块接受和输出任何数值(实数或复数)和数据类型的信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.107 所示.

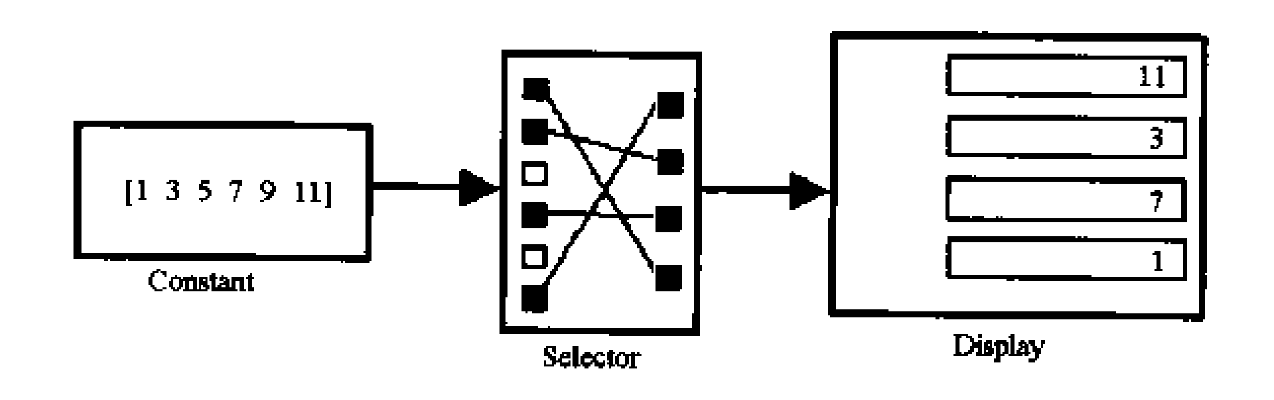


图 7.106 Selector 模块示例

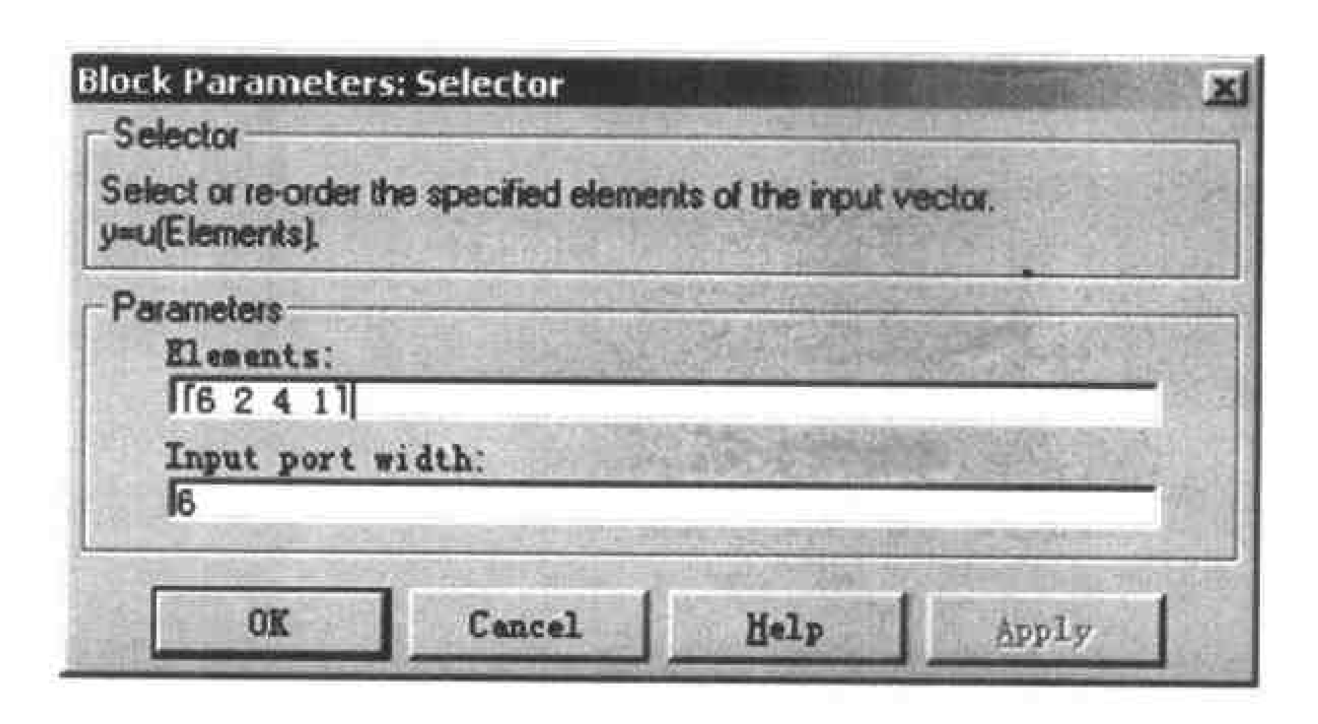


图 7.107 Selector 模块的对话框

1) 元素(Elements),在输出向量中显示的输入元素的次序.

2) 输入端口宽度(Input port width),输入向量的元素个数.

(5) 模块特点

1) 采样时间由驱动模块继承;

2) 可向量化.

7.7.22 Subsystem(子系统)

(1) 模块功能

表达一个包含在另外系统中的系统.

(2) 模块说明

Subsystem 模块表示包含在另外一个系统中的系统. 可以通过如下的方法创建子系统：

从信号与系统库中拷贝 Subsystem 模块到模型中，然后打开该 Subsystem 模块，并且拷贝模块到它的窗口中以增加模块到子系统.

用边线框选取要构成子系统的模块和连线，然后选择 Edit 菜单下的 Create Subsystem 菜单项. Simulink 用 Subsystem 模块代替这些模块. 这时打开该模块时，窗口中显示被选取的模块，并且加进了 Inport 和 Outport 模块以反映进出子系统的信号.

显示在 Subsystem 模块图标上的输入端口的个数与该子系统中 Inport 模块的个数相等. 同样，图标中显示的输出端口的个数与该子系统中 Outport 模块的个数相等. 如果没有隐藏 Inport 和 Outport 模块的名字，它们会显示为 Subsystem 模块的端口标签.

(3) 模块特点

1) 采样时间依赖子系统中的模块；

2) 向量化依赖子系统中的模块；

3) 如果有激活或触发端口，就有过零区间.

7.7.23 Terminator(终结器)

(1) 模块功能

终结未被连接的输出端口.

(2) 模块说明

Terminator 模块能够用来盖住那些输出端口没有与另外的模块相连的模块. 如果对包含有未被连接的输出端口的模型运行仿真时，Simulink 会发出警告消息. 使用 Terminator 模块盖住这些模块以避免警告消息.

(3) 模块数据类型

该模块接受任何数值和数据类型的信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.108 所示.



图 7.108　Terminator 模块参数对话框

(5) 模块特点

1) 采样时间由驱动模块继承；

2) 可向量化.

7.7.24　Trigger(触发器)

(1) 模块功能

给子系统加触发器端口.

(2) 模块说明

给子系统加 Trigger 模块,使其成为可触发的子系统.当传过触发端口的信号以给定的方式改变时,可触发子系统在每一积分步执行一次.子系统包含的 Trigger 模块不能超过一个.

可以改变 Trigger type 参数以选择触发子系统执行的事件的类型:

rising,当控制信号从 0 升到一个正数值时触发子系统执行;

falling,当控制信号从 0 降到一个负数值时触发子系统执行;

either,当控制信号从 0 升到一个正数值或者从 0 降到一个负数值时触发子系统执行;

function-call,使得子系统的执行受控于 S 函数的内部逻辑.

可以通过选取 Show output port 核选框输出触发信号.选取这一选项允许系统确定是哪类信号触发了触发器.信号的宽度是触发信号的宽度.信号值是:

1,对于导致上升沿触发的信号;

-1,对于导致下降沿触发的信号;

0,其它.

(3) 模块数据类型

该模块接受双精度或逻辑类型的信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.109 所示.

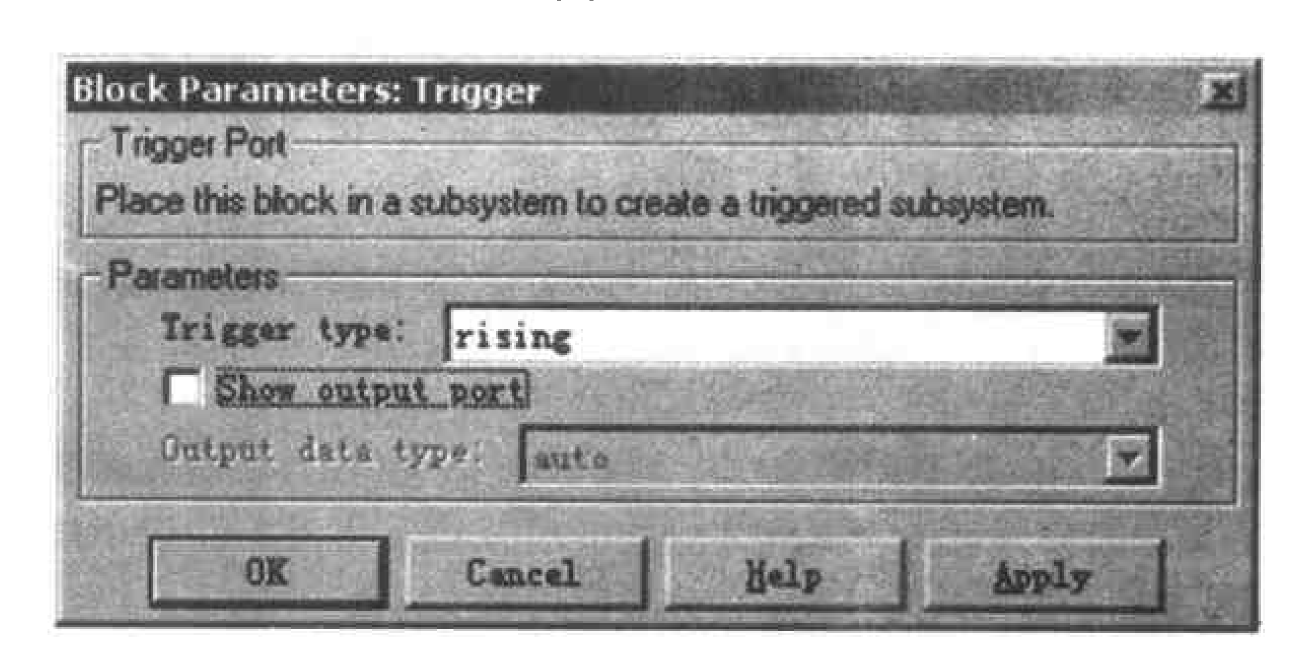


图 7.109　Trigger 模块参数对话框

1) 触发类型(Trigger type),子系统触发执行事件的类型.

2) 显示输出端口(Show output port),如果选择,将在触发模块上画一输出端口,输出触发信号.

3) 输出数据类型(Output data type),指定触发输出的数据类型(双精度或 8 位整

数). 缺省为自动(auto).

(5) 模块特点

1) 采样时间由触发端口信号确定;

2) 可向量化.

7.7.25　Width(宽度)

(1) 模块功能

输出输入向量的宽度.

(2) 模块说明

Width 模块产生的输出是其输入向量的宽度.

(3) 模块数据类型

该模块接受任何数据类型的实数或复数值信号. 输出双精度类型的实数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.110 所示.

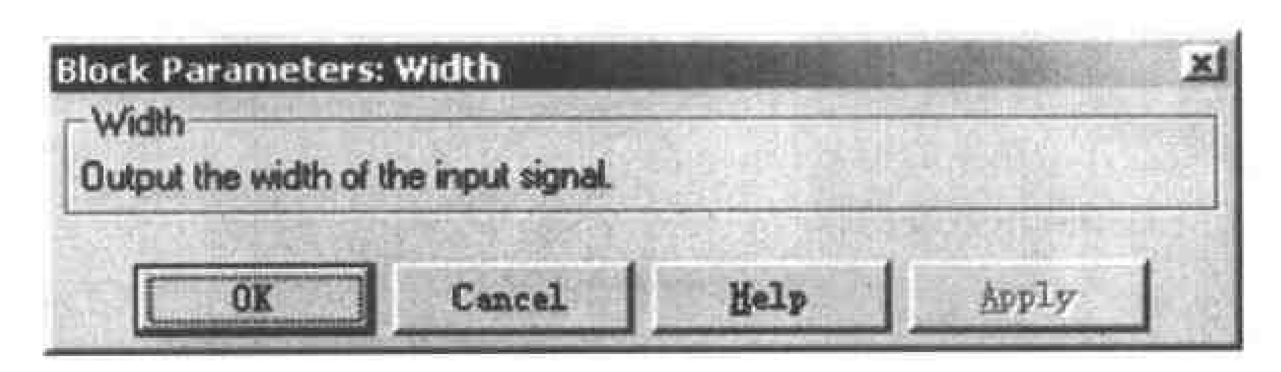


图 7.110　Width 模块参数对话框

(5) 模块特点

1) 采样时间离散;

2) 可向量化.

7.7.26　Function-Call Generator(函数调用发生器)

(1) 模块功能

以指定的速率执行函数调用子系统.

(2) 模块说明

Function-Call Generator 模块以采样时间参数指定的速率执行函数调用子系统. 要以指定顺序执行多个函数调用子系统,首先将 Function-Call Generator 模块与 Demux 模块相连,Demux 模块的输出端口数与控制的函数调用子系统数一样.

(3) 模块数据类型

该模块输出双精度类型的实数信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.111 所示.

采样时间(Sample time)参数,指采样之间的时间间隔.

(5) 模块特点

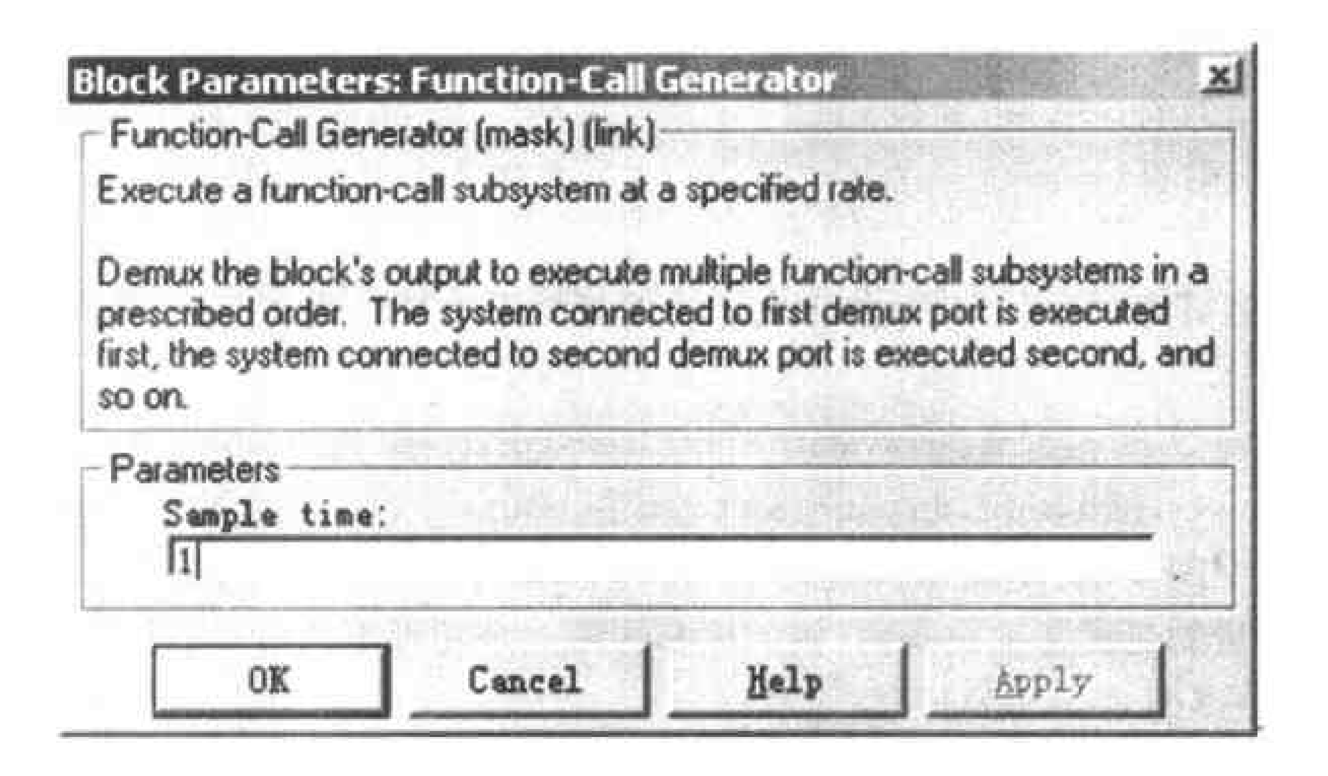


图 7.111　Function-Call Generator 模块参数对话框

1）没有直接馈通；

2）采样时间由用户指定；

3）没有标量扩展；

4）可向量化；

5）没有过零区间.

7.8　Functions & Tables 库中的模块

Functions & Tables 库中包含的模块如表 7.14 所列，各个模块的图标如图 6.5 所示.

表 7.14　Functions & Tables 库中的各模块

模块名	功　能
Fcn	对输入应用指定的表达式
Look-Up Table	实现输入的分段线性映射
Look-Up Table(2-D)	实现两个输入的分段线性映射
MATLAB Fcn	对输入应用一个 MATLAB 函数或表达式
S-Function	访问 S 函数

7.8.1　Fcn(函数表达式)

(1) 模块功能

对输入应用一个给定的表达式.

(2) 模块说明

Fcn 模块对其输入使用指定的 C 语言风格描述的表达式.表达式可以用一或多个如下元素组成：

1）u：模块的输入.如果 u 是一个向量，u(i)表示向量的第 i 个元素；u(1)或者 u 表示第一个元素.

2）数字常量.

3）算术运算符(＋,－,＊,/)

4）关系运算符(＝＝,!＝,＞,＜,＞＝,＜＝),如果表达式运算结果为真时返回 1;否则返回 0.

5）逻辑运算符(&&,||,!),如果表达式运算结果为真时返回 1;否则返回 0.

6）圆括号.

7）数学函数:abs,acos,asin,atan,atan2,ceil,cos,cosh,exp,fabs,floor,hypot,ln,log,log10,pow,power,rem,sign,sin,sinh,sqrt,tan 和 tanh.

8）工作空间变量,不能被识别为上面所列出的各项的变量名将传给 MATLAB 求值.矩阵和向量的元素必须被指明(例如矩阵的第一个元素用 A(1,1)而不是 A).

优先规则遵循 C 语言的标准:

1）()

2）＋,－(一元的)

3）pow(幂)

4）!

5）＊,/

6）＋,－

7）＞,＜,＜＝,＞＝

8）＝＝,!＝

9）&&

10）||

这里的表达式有别于 MATLAB 表达式的一个地方是它不能执行矩阵运算.另外,该模块也不支持冒号运算符(:).

模块的输入可以是标量也可以是向量.输出通常是标量.对于向量输出,考虑使用 Math Function 模块.如果模块是向量,函数运算符对输入的各个元素分别进行运算(例如 sin 函数),模块只对元素的第一个元素进行运算.

(3) 模块数据类型

该模块接受和输出双精度类型的信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.112 所示.

表达式(Expression)是对输入要应用的 C 语言风格表达式.

(5) 模块特点

1）直接馈通;

2）采样时间由驱动模块继承;

3）没有标量扩展;

4）不可向量化;

5）没有过零区间.

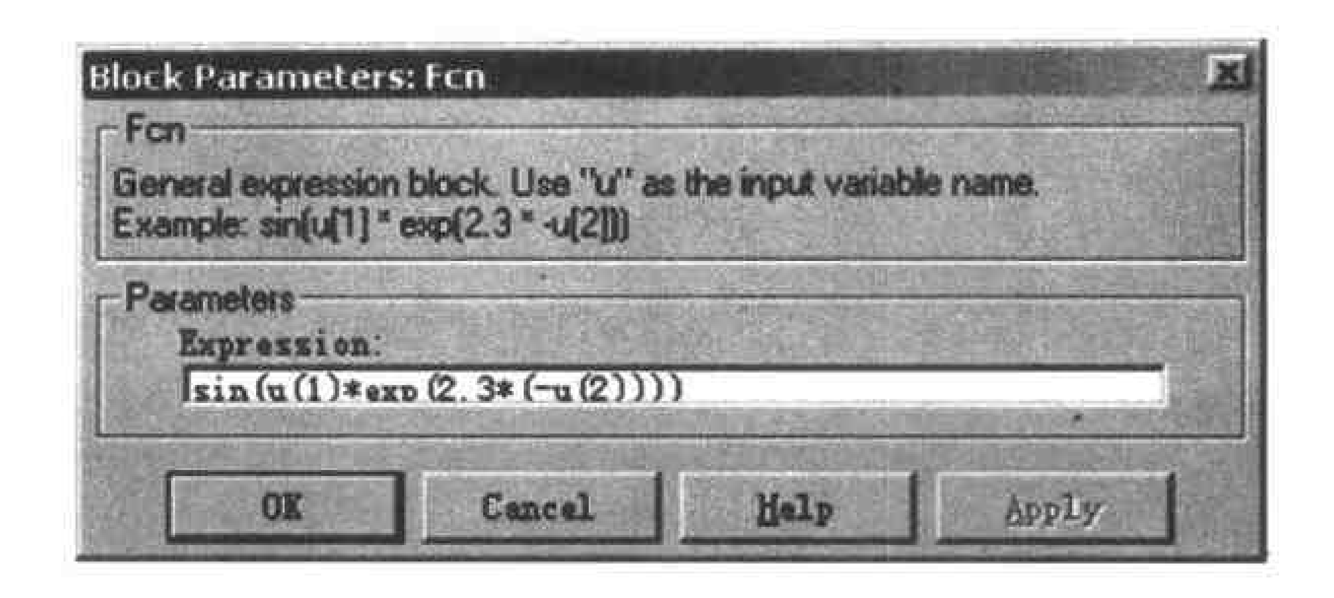


图 7.112 Fcn 模块参数对话框

7.8.2 Look-Up Table(查找表)

(1) 模块功能

执行输入的分段线性映射.

(2) 模块说明

Look-Up Table 模块使用模块参数中定义的线性插值将输入映射到输出.

可以指定(可以用行向量也可以用列向量)定义查找表的输入值向量(Vector of input values)和输出值向量(Vector of output values)参数. 模块将其输入与输入值向量中的值进行比较产生输出:

如果在输入值向量参数中找到了模块的输入值,则输出是输出值向量参数中相应的元素.

如果在输入值向量参数中没有找到模块的输入值,则在表中两个适当的元素之间进行线性插值以确定输出的值. 如果输入比输入向量参数中第一个元素还要小或者比最后一个元素还要大,模块使用开始的两个或者最后的两个元素点进行外推.

如果需要将两个输入映射到一个输出,可以使用 Look-Up Table(2-D)模块.

要创建一个阶跃的查找表,在重复输入参数的值时改变输出值.

当对于一个给定的输入值有两个点时,模块根据如下的原则产生输出:

当 u 小于 0 时,取与从原点沿负轴方向遇到的第一个点相对应的值作为输出.

当 u 大于 0 时,取与从原点沿正轴方向遇到的第一个点相对应的值作为输出.

当 u 在原点并且与零输入对应的有两个输出值,实际的输出是它们的平均值. 如果对应于零输入有三个输出值,模块产生的输出值是中间的一个.

Look-Up Table 模块的图标显示了输入向量相对于输出向量的图形. 改变模块对话框中的某一参数时,点击 Apply 或者 OK 按钮时图标自动地重画.

(3) 模块数据类型

该模块接受和输出双精度类型的信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.113 所示.

1) 输入值向量(Vector of input values),包含可能的模块的输入值. 该向量与输出向

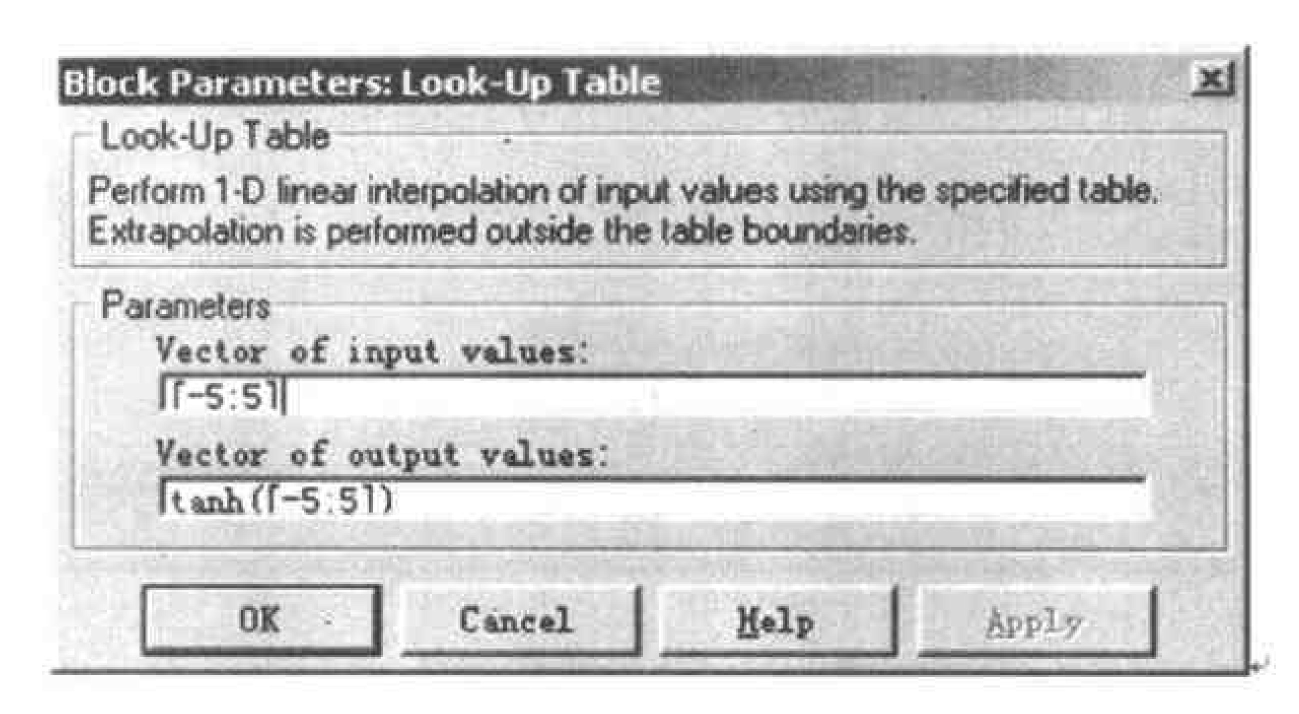


图 7.113 Look-Up Table 模块参数对话框

量必须大小一致，输入向量必须单调增加.

2）输出值向量(Vector of output values)，包含模块的输出值的向量. 与输入向量必须大小一致.

（5）模块特点

1）直接馈通；

2）采样时间由驱动模块继承；

3）没有标量扩展；

4）可向量化；

5）没有过零区间.

7.8.3 Look-Up Table(2-D)(二维查找表)

（1）模块功能

执行输入的分段线性映射.

（2）模块说明

Look-Up Table(2-D)模块使用模块参数中定义的线性插值将输入映射到输出.

可以在 Table 参数域中定义可能的输出，在行和列参数域中定义与表的行和列对应的值. 模块将其输入与行和列参数进行比较，产生其输出值. 第一个输入被认为是行的值，第二个输入被认为是列的值.

模块根据其输入值产生输出：

如果输入匹配上了行和列参数的值，输出是表中行和列相交处的值.

如果输入匹配不上行和列参数的值，模块在表中合适的值之间进行线性插值以产生输出. 如果模块至少有一个输入比行或者列参数值的第一个还要小或者比最后一个还要大，模块就根据开始或者最后两点进行外推.

如果行或者列参数之一有重复的值，模块使用 Look-Up Table 模块中介绍的方法选择一个值.

（3）模块数据类型

该模块接受和输出双精度类型的信号.

(4) 模块参数对话框

该模块的参数对话框如图 7.114 所示.

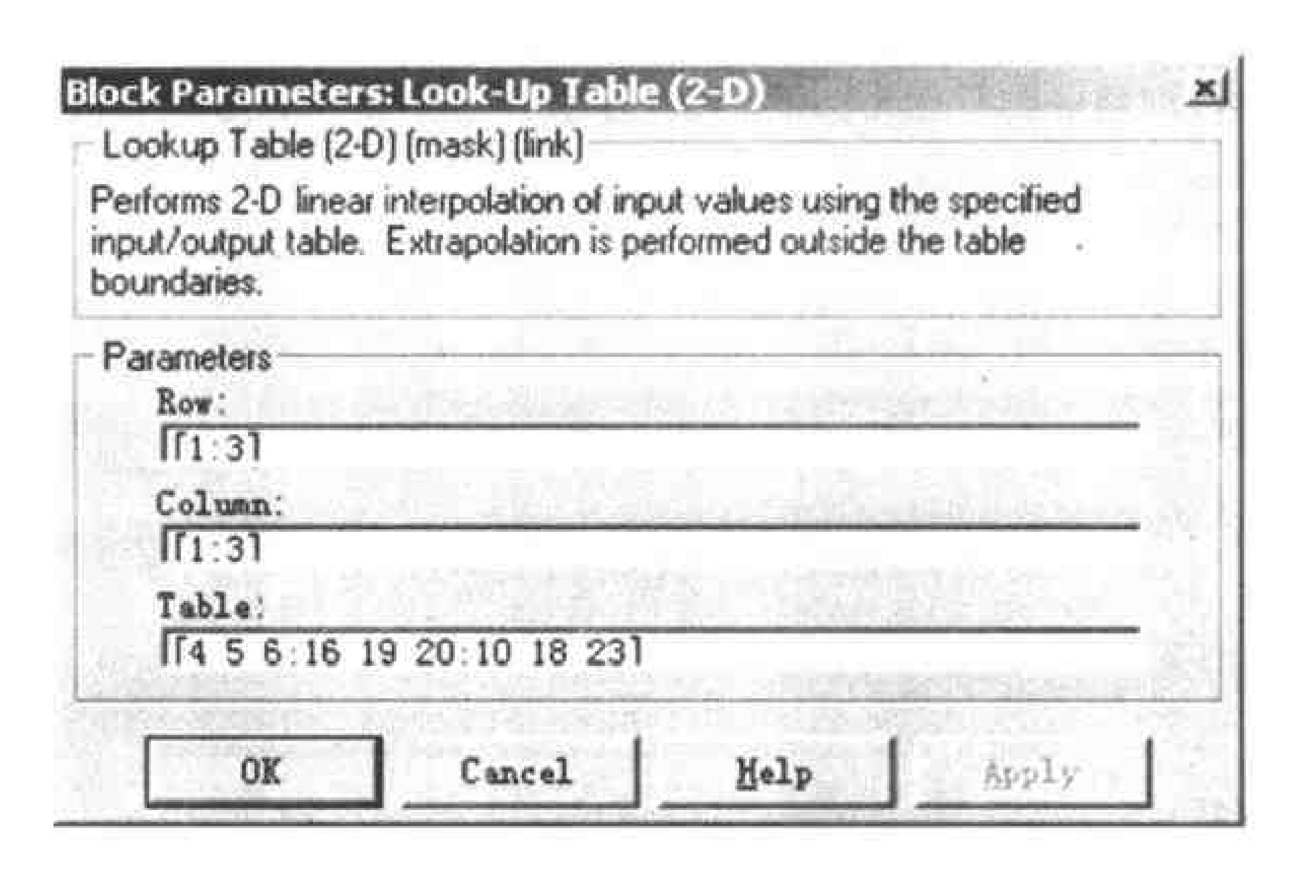


图 7.114　Look-Up Table(2-D)模块参数对话框

1) 行(Row),表的行值,以向量输入,必须单调增加.

2) 列(Column),表的列值,以向量输入,必须单调增加.

3) 表(Table),输出值表,矩阵大小必须与行列参数定义的维数相匹配.

(5) 模块特点

1) 直接馈通;

2) 采样时间由驱动模块继承;

3) 有标量扩展;

4) 可向量化;

5) 没有过零区间.

7.8.4　MATLAB Fcn(MATLAB 函数)

(1) 模块功能

对输入使用 MATLAB 函数或者表达式.

(2) 模块说明

MATLAB Fcn 模块对其输入应用指定的 MATLAB 函数或者表达式. 模块接收一个输入并且产生一个输出. 指定的函数或者表达式被用于输入. 函数的输出数目必须与模块的输出数目一致,否则会出现错误.

下面是对于该模块来说一些有效的表达式的例子:

```
sin
atan2(u(1), u(2))
u(1)^ u(2)
```

该模块比 Fcn 模块要慢一些,因为它在每一积分步调用 MATLAB 的分析器. 可以考

虑使用内建模块(如 Fcn 模块或者 Math Function 模块)来代替它,或者将该函数写成 M 文件形式或者 MEX 文件形式的 S 函数,然后使用 S-Function 模块访问它.

(3) 模块数据类型

该模块接受双精度类型的实数或复数值信号,并产生双精度类型的实数或复数输出,但要依赖输出信号类型的参数.

(4) 模块参数对话框

该模块的参数对话框如图 7.115 所示.

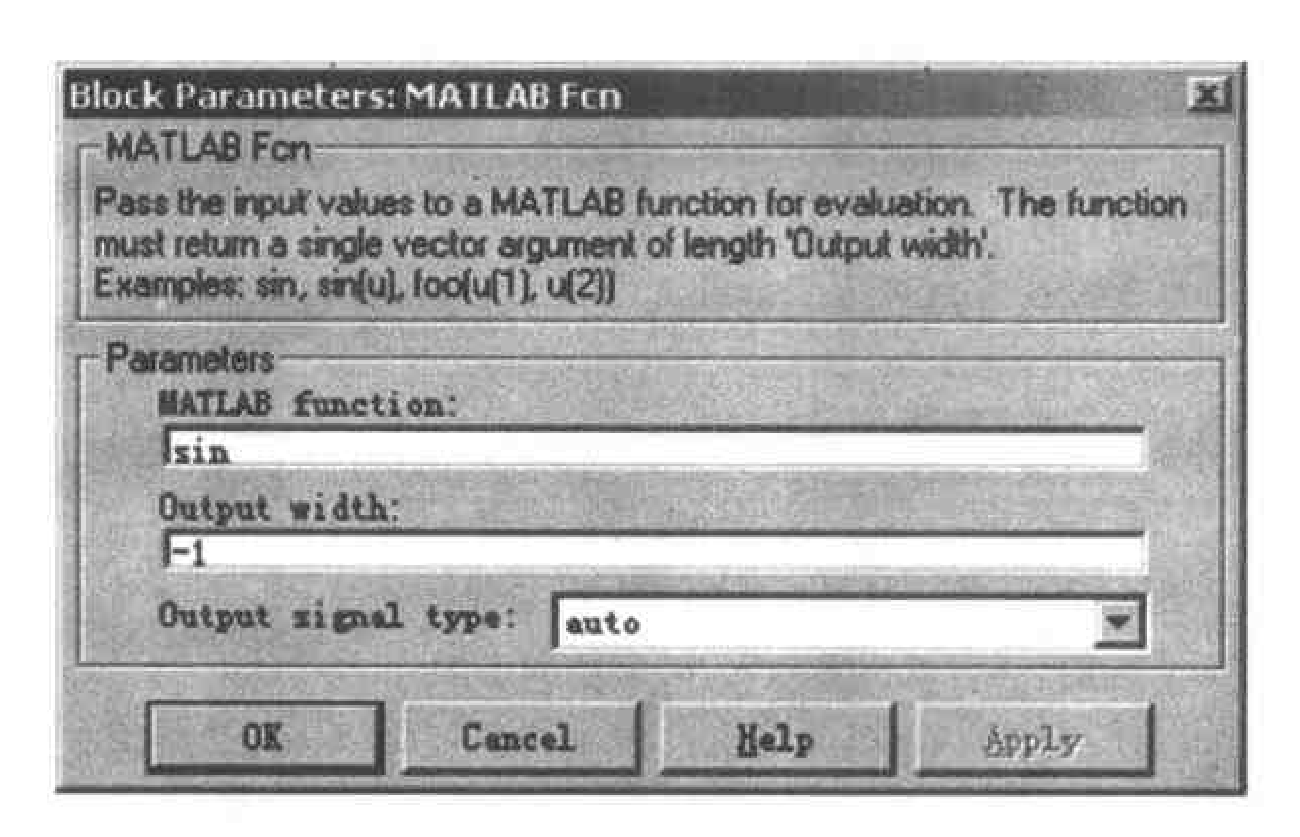


图 7.115　MATLAB Fcn 模块参数对话框

1) MATLAB 函数,指函数或表达式.如果只指定为函数,就不必包含输入参数.

2) 输出宽度(Output width),如果输出与输入宽度一致,则指定为−1.

3) 输出信号类型(Output signal type),可选择自动(auto),复数(complex)或实数(real).

(5) 模块特点

1) 直接馈通;

2) 采样时间由驱动模块继承;

3) 标量扩展 N/A;

4) 可向量化;

5) 没有过零区间.

7.8.5　S-Function(S 函数)

(1) 模块功能

访问 S 函数.

(2) 模块说明

S-Function 模块提供从模块图中访问 S 函数的方法.S-function name 参数中 S 函数可以是 S 函数的 M 文件或者 MEX 文件.

S-Function 模块允许直接传递附加参数给 S 函数.函数的参数可以指定为 MATLAB

的表达式或者是用逗号分开的变量. 例如:

A, B, C, D, [eye(2, 2); zeros(2, 2)]

尽管单个的参数可以用方括号括起来,但是参数列表不能用方括号括起来.

S-function 模块的图标显示指定的 S 函数的名字并且不管它包含的子系统有多少个输入和输出,通常只画一个输入端口和一个输出端口.

当 S 函数包含有多个输入或者多个输出时使用向量信号线. 输入向量的宽度必须与包含在 S 函数中的输入的数目相等. 模块将输入向量的第一个元素传给 S 函数的第一个输入,将输入向量的第二个元素传给 S 函数的第二个输入,如此等等. 同样,输出向量的宽度必须与 S 函数输出的数目相等.

(3) 模块数据类型

依赖 S 函数模块的执行.

(4) 模块参数对话框

该模块的参数对话框如图 7.116 所示.

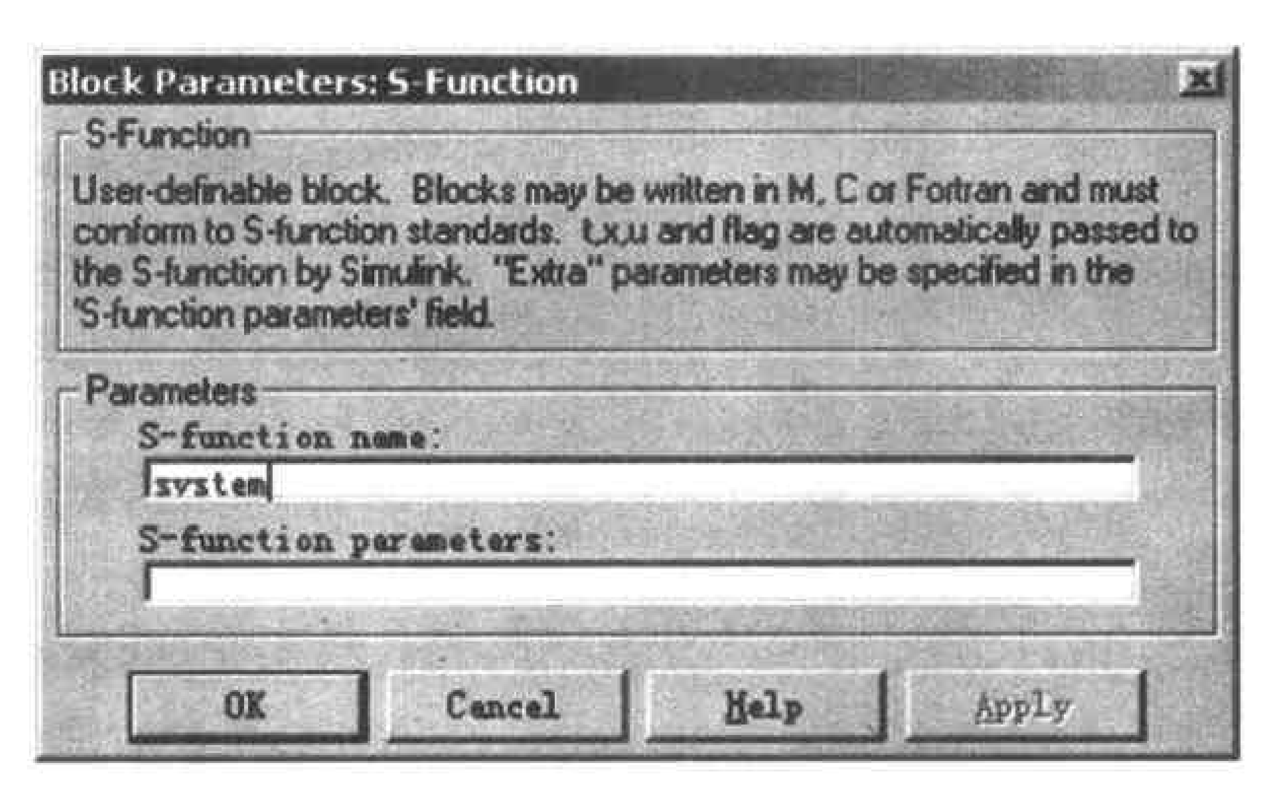


图 7.116　S-function 模块参数对话框

参数包括:S 函数名和 S 函数参量.

(5) 模块特点

1) 直接馈通依据 S 函数的内容;

2) 采样时间依赖 S 函数内容;

3) 标量扩展依赖 S 函数内容;

4) 向量化依赖 S 函数内容;

5) 没有过零区间.

第八章　模型创建与调试命令

8.1　如何指定 Simulink 对象路径

下面描述的命令需要标识一个 Simulink 系统或者模块. 通过指定它们的路径来标识系统和模块：

1）要标识一个系统，指定系统的名字，也就是包含系统描述的文件的名字，不要带 mdl 扩展名.

2）要标识一个子系统，指定该子系统所处的系统及子系统层次：

system/subsystem1/. . . /subsystem

3）要标识一个模块，指定包含该模块的系统的路径及模块的名字：

system/subsystem1/. . . /subsystem/block

如果模块的名字包括回车或者换行符，指定该模块的名字为一个字符串并且使用 sprintf('\n') 作为换行符. 例如，下面这些命令行将换行符赋给 cr，然后取得 Signal Generator 的 Amplitude 参数的值：

```
cr=sprintf('/n');
get_param(['untitled/Signal', cr , 'Generator'], 'Amplitude')
ans=
    1
```

如果模块的名字包含有斜杠(/)，当指定模块的名字时应该输入两个斜杠. 例如，要取得 mymodel 系统中名为 Signal/Noise 模块的 Location 参数的值，命令输入的方法如下：

```
get_param('mymodel/Signal//Noise', 'Location')
```

8.2　模型创建命令

表 8.1 简要介绍了该章要详细说明的模型创建命令及其所执行的主要任务.

8.2.1　add_block 命令

功能：加模块到 Simulink 系统.

语法：add_block('src', 'dest')

add_block('src', 'dest', 'parameter1', value1, . . .)

说明：

add_block('src', 'dest')，拷贝一个包含有完整路径名称'src'的模块到系统，并指明新的模块的完整路径名'dest'. 新的模块的参数与原来的模块的参数相同. 可以用'build.in'作为所有 Simulink 内建模块(Simulink 模块库中可用的并且不是模板的模块)的源系统名

字.

add_block('src','dest','parameter1',value1,...),创建如前面所说的一个拷贝,但其被指明的参数具有被指定的值.任何另外指明的参数值必须参数和值成对地出现.

表 8.1　创建模型的命令

命　令	说　明
new_system	创建新 Simulink 的系统
open_system	打开已存在的系统
close_system, bdclose	关闭某一系统的窗口
save_system	保存 Simulink 系统
find_system	查找 Simulink 系统、模块、线、注释等
add_block	向系统中加入新的模块
delete_block	从系统中删除模块
replace_block	在系统中替换模块
add_line	加入连线到系统中
delete_line	从系统中删除连线
get_param	获取参数值
set_param	设置参数值
gcb	获取当前模块的路径
gcs	获取当前系统的路径
gcbh	获得当前模块句柄
bdroot	获取根层系统的名字
simulink	打开 Simulink 模块库

例 8.1　下面的命令从 simulink 系统的 Sinks 子系统拷贝 Scope 模块到 engine 系统的 timing 子系统中,名字换成 Scope1:

add_block('simulink/Sinks/Scope','engine/timing/Scope1')

下面的命令在 F14 系统中创建一个名为 controller 的新的子系统:

add_block('built_in/SubSystem','F14/controller')

下面的命令拷贝内建的 Gain 模块到 example81 系统,新的模块的名字为 Volume,并且指定它的 Gain 参数值为 4:

add_block('built_in/Gain','example81/Volume','Gain','4')

参见:delete_block,set_param 命令

8.2.2　add_line 命令

功能:加连线到 Simulink 系统中.

语法:h=add_line('sys','oport','iport')

h=add_line('sys',points)

说明:

add_line 命令在指定的系统中增加一条连线.这条连线有两种定义方法：

1）指明要被连线连接的端口的名字.

2）指定定义连线线段的端点的位置.

add_line('sys', 'oport', 'iport'),在系统中增加一条从指定的模块输出端口'oport'到指定的模块输入端口'iport'的连线.'oport'和'iport'是由模块名字和端口标识以'block/port'的形式组成的字符串.大多数模块端口是通过从顶层到底层从左到右的顺序编号加以标识的,如'Gain/1'或者'Sum/2'. Enable(激活)、Trigger(触发)和 State(状态)端口是通过名字加以标识,如'subsystem_name/Enable'、'subsystem_name/Trigger'或者'Integrator/State'.

add_line('sys', points),增加线段到一个系统.数组 points 的第一行指定线段端点的 x 和 y 坐标.原点是窗口的左上角.信号从第一行定义的点出发传到最后一行定义的点.如果新的连线的起点靠近一个已存在的模块的输出或者一条连线,它们之间将会相连.同样,如果连线的末端靠近一个已经存在的输入,它们之间也将会连接起来.

例 8.2 下面的命令在 example81 系统中增加一条连线以连接 Sine Wave 模块的输出到 Mux 模块的第一个输入：

add_line('example81', 'Sine Wave/1', 'Mux/1')

下面的命令在 example81 中增加一条连线,该连线分成两段,第一段从(20,20)到(40,10),第二段从(40,10)到(60,60)：

add_line('example81', [20 20; 40 10; 60 60])

参见:delete_line 命令

8.2.3 bdclose 命令

功能:无条件地关闭 Simulink 系统窗口.

语法:bdclose

bdclose('sys')

bdclose('all')

说明：

不带参数的 bdclose 命令无条件且不经确认地关闭当前系统的窗口.该系统从最后一次保存到关闭时所做的修改都将丢失.

bdclose('sys'),关闭指定系统的窗口.

bdclose('all'),关闭所有系统的窗口.

例 8.3 下面的命令关闭 vdp 系统：

bdclose('vdp')

参见:close_system,new_system,open_system,save_system 命令

8.2.4 bdroot 命令

功能:返回顶层 Simulink 系统的名字.

语法:bdroot

bdroot('obj')

说明：

不带参数的 bdroot 命令返回顶层系统的名字.

bdroot('obj')，返回包含指定对象名字的顶层系统的名字，其中'obj'是系统或者模块路径的名字.

例 8.4　下面的命令返回当前模块的顶层系统的名字：

bdroot(gcb)

参见：find_system，gcb 命令

8.2.5　close_system 命令

功能：关闭 Simulink 系统的窗口或者模块的对话框.

语法：close_system

close_system('sys')

close_system('sys', saveflag)

close_system('sys', 'newname')

close_system('blk')

说明：

不带参数的 close_system 命令关闭当前系统或者子系统的窗口.如果当前系统是顶层系统并且它被修改过，那么该命令在将系统从内存清除之前询问是否将修改过的系统保存到文件.当前系统由 gcs 命令定义.

close_system('sys')，关闭指定的'sys'系统或者子系统窗口.

close_system('sys', saveflag)关闭指定的顶层系统窗口并且将它从内存清除掉：

如果 saveflag 是 0，系统不保存.

如果 saveflag 是 1，系统用当前的名字保存.

close_system('sys', 'newname')，用指定的新名字保存指定的顶层系统到文件，然后关闭系统.

close_system('blk')关闭与指定的模块关联的对话框，或者如果定义了 CloseFcn 回调函数，就调用模块的 CloseFcn 函数，其中'blk'是完整的模块路径名称.任何另外的参数都将忽略.

例 8.5　下面的命令关闭当前的系统：

close_system

下面的命令关闭 vdp 系统：

close_system('vdp')

下面的命令用它当前的名称保存 engine 系统，然后关闭它：

close_system('engine', 1)

下面的命令用 testsys 的名字保存 example81 系统，然后关闭它：

close_system('example81', 'testsys')

下面的命令关闭 engine 系统的 Combustion 子系统中的 Unit Delay 模块的对话框：

close_system('engine/Combustion/Unit Delay')

参见：bdclose，gcs，new_system，open_system，save_system 命令

8.2.6 delete_block 命令

功能：从 Simulink 系统中删除一个模块.

语法：delete_block('blk')

说明：

delete_block('blk')，从系统中删除指定的模块.其中'blk'是完整的模块路径名.

例 8.6 下面的命令从 vdp 系统中删除 Out1 模块：

delete_block('vdp/Out1')

参见：add_block 命令

8.2.7 delete_line 命令

功能：从 Simulink 系统中删除连线.

语法：delete_line('sys', 'oport', 'iport')

delete_line('sys', [x y])

说明：

delete_line('sys', 'oport', 'iport')，删除从指定的模块输出端口'oport'到指定的模块输入端口'iport'的连线.'oport'和'iport'是由模块名字和端口标识以'block/port'的形式组成的字符串.大多数模块端口是通过从顶层到底层从左到右的顺序编号加以标识的，如'Gain/1'或者'Sum/2'. Enable(使能)、Trigger(触发)和 State(状态)端口是通过名字加以标识，如'subsystem_name/Enable'、'subsystem_name/Trigger'或者'Integrator/State'.

delete_line('sys', [x y])，删除系统中包含有指定点(x, y)的连线(如果这样的连线存在).

例 8.7 下面的命令删除 example81 系统中连接 Sum 模块与 Mux 模块第二个输入的连线：

delete_line('example81', 'Sum/1', 'Mux/2')

参见：add_line 命令

8.2.8 find_system 命令

功能：用指定的参数值查找 Simulink 的对象.

语法：find_system(sys, constraint, cv 'parameter1', value1, 'parameter2', value2, ...)

说明：

find_system(sys, constraint, cv 'parameter1', value1, 'parameter2', value2, ...)，搜索由 sys 指定的系统或子系统，同时使用 constraint 指定的约束，返回指定参数值 value1，value2 等的对象路径或句柄. sys 可以是一路径(路径名单元数组)，一个句柄(或句柄向量)，也可以省掉.如果 sys 为路径名或路径名单元数组，该命令则返回所发现对象的路径名单元数组；如果 sys 为一个句柄或句柄向量，则返回其发现的句柄向量；如果省掉 sys，则搜索所有打开系统.

参数名字忽略大小写，而值字符串则是区分大小写的.

搜索约束可指定为表 8.2 中的任何一个.

表 8.2　搜索约束

名　称	值 类 型	说　明
'SearchDepth'	标量	约束搜索深度为指定层，0 为只搜索打开的系统，1 为搜索顶层系统的模块或子系统，2 为搜索顶层子系统及其下层，缺省时搜索所有系统.
'LookUnderMasks'	'on'或'off'	如果为'on'则扩展到模板系统，缺省为'off'.
'FollowLinks'	'on'或'off'	如果为'on'搜索沿连接的库模块，缺省为'off'.
'FindAll'	'on'或'off'	如果为'on'搜索扩展到连接线和注释，缺省为'off'.

如果约束参数省掉，则使用缺省的约束.

find_system('SearchDepth', depth, 'parameter1', value1, ...)限制从顶层系统搜索到指定的深度. 标量 depth 参数是：

例 8.8　下面的命令返回一个包含有所有打开的系统和模块的名字的单元数组：

```
find_system
```

下面的命令返回所有打开的模块图的名字：

```
open_bd=find_system('Type', 'block_diagram')
```

下面的命令返回 clutch 系统的 Unlocked 子系统中的所有 Goto 模块：

```
find_system('clutch/Unlocked', 'SearchDepth', 1, 'BlockType', 'Goto')
```

下面这些命令返回 vdp 系统中所有 Gain 参数值为 1 的 Gain 模块：

```
gb=find_system('vdp', 'BlockType', 'Gain')
find_system(gb, 'Gain', '1')
```

上面的命令等价于下面的命令：

```
find_system('vdp', 'BlockType', 'Gain', 'Gain', '1')
```

参见：get_param，set_param 命令

8.2.9　gcb 命令

功能：获取当前 Simulink 模块的完整路径名.

语法：gcb

```
gcb('sys')
```

说明：

gcb，返回当前系统中当前模块的完整路径名.

gcb('sys')，返回指定系统中当前模块的完整路径名.

当前模块是下面所列中的一种：

1) 在编辑时，当前模块是最近点击的模块.

2) 在运行一个包含有 S-Function 模块的系统的仿真时，当前模块是当前正在执行其相应的 MATLAB 函数的 S-Function 模块.

3) 在回调时，当前模块是那个正在执行它的回调程序的模块.

4) 在计算 MaskInitialiazation 字符串时，当前模块是那个正在计算它的模板的模块.

例 8.9 下面的命令返回最近被选取的模块的路径：

```
gcb
ans=
    clutch/Locked/Inertia
```

下面的命令取得当前模块的 Gain 参数的值：

```
get_param(gcb, 'Gain')
ans=
    1/(Iv+Ie)
```

参见：gcbh，gcs 命令

8.2.10 gcbh 命令

功能：取得当前 Simulink 模块的句柄.

语法：gcbh

说明：

gcbh 返回当前系统中当前模块的句柄.

可以使用该命令标识没有父系统的模块或者给出它的地址. 这一命令对于模块集的作者非常有用.

例 8.10 下面的命令返回最近被选取的模块的句柄：

```
gcbh
ans=
    5.0004
```

参见：gcb 命令

8.2.11 gcs 命令

功能：取得当前 Simulink 系统的完整路径名.

语法：gcs

说明：

gcs 返回当前系统的完整路径名.

当前系统是：

1）在编辑时，当前系统是最近在其中点击过的系统.

2）在运行一个包含有 S-Function 模块的系统的仿真时，当前系统是包含当前正在计算的 S-Function 模块的系统或者子系统.

3）在回调时，当前系统是包含正在执行其回调程序的模块的系统.

4）在计算 MaskInitialiazation 字符串时，当前系统是包含正在计算其模板的模块的系统.

例 8.11 下面的例子返回包含最近被选取的模块的系统的路径：

```
gcs
ans =
example81
```

参见:gcb 命令

8.2.12　get_param 命令

功能:取得 Simulink 的系统或者模块的参数的值.

语法:get_param('obj', 'parameter')

get_param({objects}, 'parameter')

get_param(handle, 'parameter')

get_param('obj', 'ObjectParameter')

get_param({objects}, 'DialogParameter')

说明:

get_param('obj', 'parameter'),返回指定参数的值,其中'obj'是系统或者模块的路径名.忽略参数的大小写.

get_param({objects}, 'parameter'),接受一个指定完整路径的单元数组,这样能够取得在单元数组中指定的所有对象所共有的参数的值.

get_param(handle, 'parameter'),返回用 handle 指定句柄的对象的指定参数值.

get_param('obj', 'ObjectParameter'),返回描述 obj 参数的结构.返回结构的每个字段与特定的参数相对应,并具有一个参数名.每个参数字段又包括三个字段:名字、类型和属性.

get_param({objects}, 'DialogParameter'),返回一个包含指定模块对话参数的名字的单元数组.

例 8.12　下面的命令返回 clutch 系统的 Requisite Friction 子系统中的 Inertia 模块的 Gain 参数的值:

```
get_param('clutch/Requisite Friction/Inertia', 'Gain')
ans=
1/(Iv+Ie)
```

下面这些命令显示当前系统(mx+b 系统)中所有模块的模块类型:

```
blks=find_system(gcs, 'Type', 'block');
listblks=get_param(blks, 'BlockType')
listblks=
    'SubSystem'
    'Inport'
    'Constant'
    'Gain'
    'Sum'
    'Outport'
```

参见:find_system,set_param 命令

8.2.13　new_system 命令

功能:创建一个空的 Simulink 系统.

语法:new_system('sys')

说明:

new_system('sys'),用指定的名字创建一个新的空的系统.如果'sys'指定的是路径,那么将会在指定的路径下的系统中创建一个新的子系统. new_system 不打开系统的窗口.

例 8.13 下面的命令创建一个名为'newsys'的新的系统:

new_system('newsys')

下面的命令在 vdp 系统中创建一个名为'newsys'的新的子系统:

new_system('vdp/newsys')

参见:close_system,open_system,save_system 命令

8.2.14 open_system 命令

功能:打开 Simulink 系统的窗口或者模块的对话框.

语法:open_system('sys')

open_system('blk')

open_system('blk','force')

说明:

open_system('sys'),打开指定的系统或者子系统窗口.

open_system('blk'),打开与指定的模块相关联的对话框,其中'blk'是完整的模块路径名.如果定义了模块的 OpenFcn 回调函数,计算该程序.

open_system('blk','force'),查看指定系统的模板内的情况,其中'blk'是完整路径名或者模板系统.该命令等价于使用 Look Under Mask 菜单项.

例 8.14 下面的命令在缺省的显示位置打开 controller 系统:

open_system('controller')

下面的命令打开 controller 系统内的 Gain 模块的对话框:

open_system('controller/Gain')

参见:close_system,new_system,save_system 命令

8.2.15 replace_block 命令

功能:在 Simulink 模型中替换模块.

语法:replace_block('sys','blk1','blk2','noprompt')

replace_block('sys','Parameter','value','blk',...)

说明:

replace_block('sys','blk1','blk2','noprompt'),用 'blk2'替换 'sys'系统中所有类型为 'blk1'的模块或者模板.如果 'blk2'是 Simulink 的内建模块,只需要指明模块的名字.如果 'blk2'在另外的系统中,需要指明完整的模块路径名.如果没有带'noprompt'参数,Simulink 在替换之前显示一个对话框要求选择匹配的模块.指定 'noprompt'参数就不会出现该对话框.如果指定返回变量,被替换的模块的路径保存在那个变量中.

replace_block('sys', 'Parameter', 'value', 'blk', ...),用 'blk'模块替换 'sys'系统中所有指定的参数具有指定值的模块.可以指定任何数目的参数/值对.

例 8.15 下面的命令用 Integrator 模块替换 f14 系统中的所有 Gain 模块并且将被替换模块的路径保存在 RepNames 变量中.Simulink 在替换之前列出匹配的模块:

RepNames=replace_block('f14', 'Gain', 'Integrator')

下面的命令替换用 Integrator 模块替换 clutch 系统的 Unlocked 子系统中所有 Gain 参数为'bv'的模块.Simulink 在替换之前显示一个对话框列出匹配的模块:

replace_block('clutch/Unlocked', 'Gain', 'bv', 'Integrator')

下面的命令用 Integrator 模块替换 f14 系统中的所有 Gain 模块,不显示对话框:

replace_block('f14', 'Gain', 'Integrator', 'noprompt')

参见:find_system,set_param 命令

8.2.16 save_system 命令

功能:保存 Simulink 系统.

语法:save_system

save_system('sys')

save_system('sys', 'newname')

说明:

save_system,用当前的名字保存当前的顶层系统到文件中.

save_system('sys'),用当前的名字保存指定的顶层系统到文件.该系统必须已经打开.

save_system('sys', 'newname'),用指定的新的名字保存指定的顶层系统到文件.该系统必须已经打开.

例 8.16 下面的命令保存当前系统:

save_system

下面的命令保存 vdp 系统:

save_system('vdp')

下面的命令用'myvdp'的名字保存 vdp 系统到文件:

save_system('vdp', 'myvdp')

参见:close_system,new_system,open_system 命令

8.2.17 set_param 命令

功能:设置 Simulink 系统和模块参数.

语法:set_param('obj', 'parameter1', value1, 'parameter2', value2, ...)

说明:

set_param('obj', 'parameter1', value1, 'parameter2', value2, ...),用指定的值设置指定的参数,其中 'obj'是系统或模块的路径.忽略参数名的大小写、参数值字符串区分大小写.任何与对话框的条目相应的参数具有字符串形式的值.

可以在仿真期间在工作间中改变模块参数的值，并且用这些改动更新模块图. 要做到这一点，在命令窗口中做这些改动，接着使模型窗口成为活动窗口，然后从 Edit 菜单中选择 Update Diagram 菜单项.

大多数模块参数值必须以字符串的形式指定，除了所有模块共有的 Position 和 UserData 参数以外.

例 8.17 下面的命令设置 vdp 系统的 Solver 和 StopTime 参数：

```
set_param('vdp', 'Solver', 'ode15s', 'StopTime', '3000')
```

下面的命令设置 vdp 系统中 Mu 模块的 Gain 为 1000：

```
set_param('vdp/Mu', 'Gain', 1000)
```

下面的命令设置 vdp 系统中 Fcn 模块的位置：

```
set_param('vdp/Fcn', 'Position', [50 100 110 120])
```

下面的命令设置 mymodel 系统中 Zero-Pole 模块的 Zeros 和 Poles 参数：

```
set_param('mymodel/Zero-Pole', 'Zeros', '[2 4]', 'Poles', '[1 2 3]')
```

下面的命令设置模板子系统中的一个模块的 Gain 参数，变量 k 与 Gain 参数相关联：

```
set_param('mymodel/Subsystem', 'k', '10')
```

下面的命令设置 mymodel 系统中名为 Compute 的模块的 OpenFcn 回调参数. 当用户双击 Compute 模块时执行 'my_open_fcn' 函数.

```
set_param('mymodel/Compute', 'OpenFcn', 'my_open_fcn')
```

参见：get_param，find_system 命令

8.2.18 simulink 命令

功能：打开 Simulink 模块库.

语法：simulink

说明：

simulink 命令打开 Simulink 模块库窗口，创建且显示一个新的空模型窗口，下列两种情况除外：

1) 如果已经打开了一个模型窗口，Simulink 不创建新的模型窗口.

2) 如果已经打开了 Simulink 模块库窗口，发布该条命令使 Simulink 窗口成为活动窗口.

8.3 模型调试命令

Simulink 调试器是一个定位和诊断模型错误的工具. 可以通过一步步运行并显示模块的状态、输入和输出来查清问题的所在. 使用 sldebug 命令或 sim 命令的调试选项来打开一个模型，使其处于调试控制之下.

例 8.18 输入如下命令

```
sldebug 'example53'
[Tm=0                    ] * * Start * * of system 'example53' outputs
(sldebug @0:0 'example53/反馈')：
```

或命令：

sim('example53',[0,10],simset('debug','on'))

sim('example53',[0,10],simset('debug','on'))

[Tm=0] * * Start * * of system 'example53' outputs

(sldebug @0:0 'example53/反馈')：

表 8.3 列出了 Simulink 的调试命令.

表 8.3 Simulink 调试命令

命　令	缩　写	说　明
ashow	as	显示一个代数回路
atrace	at	设置代数回路跟踪级
bafter	ba	在一个模型执行后插入一个断点
break	b	在一个模型执行前插入一个断点
bshow	bs	显示一个指定模块
clear	cl	从一个模块中清除一个断点
continue	c	继续仿真
disp	d	当仿真停止时显示模块的输入/输出
help	? 或 h	显示调试命令的帮助
ishow	i	允许或不允许显示积分信息
minor	m	允许或不允许最小步模式
nanbreak	na	设置或清除非限定值中断
next	n	从下一时间步开始
probe	p	显示一个模块的输入/输出
quit	q	中止仿真
run	r	运行仿真直到完成
slist	sli	列一模型的非虚拟模块
states	state	显示当前状态值
status	stat	显示有效的调试选项
step	s	跳至下一个模块
stop	sto	停止仿真
systems	sys	列出一个模型的非虚拟系统
tbreak	tb	设置或清除一个时间断点
trace	tr	每次执行显示模块的输入/输出
undisp	und	从调试显示点列表中去除一个模块
untrace	unt	从调试跟踪点列表中去除一个模块
xbreak	x	当调试器遇到步长极限状态时中断
zcbreak	zcb	在非采样过零事件时中断
zclist	zcl	列出包含非采样过零模块

8.3.1 ashow 命令

功能：显示一个代数回路.

语法：ashow <gcb | s:b | s#n | clear>

说明：

ashow gcb，突出当前模块中的代数回路.

ashow s:b，突出指定系统 s 中的第 n 个代数回路.

ashow s#n，突出系统 s 中第 n 个代数回路.

ashow clear，清除模型图中突出显示的代数回路.

参见：atrace，slist 命令

8.3.2 atrace 命令

功能：设置代数回路跟踪级.

语法：atrace level

说明：

该命令设置仿真代数回路跟踪级.

atrace 0，不显示任何信息.

atrace 1，显示循环变量结果，需要给出迭代次数和估计结果误差.

atrace 2，与 atrace 1 一样.

atrace 3，2 级加 Jacobi 矩阵用于解循环.

atrace 4，3 级加循环变量的中间结果.

参见：systems，states 命令

8.3.3 bafter 命令

功能：在模块执行后插入一个中断点.

语法：bafter gcb

　　　bafter s:b

说明：

bafter gcb，在当前模块执行后，插入一个中断点.

bafter s:b，在系统索引为 s，模块索引为 b 的模块执行后，插入一个中断点.

参见：break，xbreak，tbreak，nanbreak，zcbreak，slist 命令

8.3.4 break 命令

功能：在一个模块执行前，插入一个中断点.

语法：break gcb

　　　break s:b

说明：

break gcb，在当前模块执行前，插入一个中断点.

break s:b，在系统索引为 s、模块索引为 b 的模块执行前，插入一个中断点.

参见:bafter,xbreak,tbreak,nanbreak,zcbreak,slist 命令

8.3.5 bshow 命令

功能:显示一个指定的模块.

语法:bshow s:b

说明:

bshow s:b,打开包模型窗口,其中含有系统索引为 s,模块索引为 b 的模块.

参见:slist 命令

8.3.6 clear 命令

功能:从一个模块清除中断点.

语法:clear gcb

　　clear s:b

说明:

clear gcb 和 clear s:b,清除指定模块的中断点.参数含义与前面一样.

参见:bafter,slist 命令

8.3.7 continue 命令

功能:继续仿真.

语法:continue

说明:

该命令从当前中断点继续执行仿真.仿真执行一直到下一个中断点或最终时间步.

参见:run,stop,quit 命令

8.3.8 disp 命令

功能:当仿真停止时,显示一个模块的输入和输出.

语法:disp

　　disp gcb

　　disp s:b

说明:

disp gcb 和 disp s:b 命令将一个指定模块注册为显示点.当仿真中止时,调试器在 MATLAB 命令行中显示所有显示点的输入和输出.

不带任何参数的 disp 命令显示所有显示点列表.

使用 undisp 解除一个模块显示点.

参见:undisp,slist,probe,trace 命令

8.3.9 help 命令

功能:显示调试器命令的帮助.

语法:help

说明：

该命令在命令窗口显示调试器命令列表. 列表中包含每个命令的语法和简单说明.

例 8.19　在例 8.18 运行结果下输入？或 h 或 help 得到：

```
sldebug 'example53'
[Tm=0]                  * * Start * * of system 'example53' outputs
(sldebug @0:0 'example53/反馈'): help
Commands:
step                    Step to next block.
next                    Go to start of next time step.
disp [s:b | gcb]        Register or display I/O of block(s)at every stopping point.
undisp <s:b | gcb>      Remove a display point.
trace <s:b | gcb>       Add a trace point to display block I/O as it is executed.
untrace <s:b | gcb>     Remove a trace point.
probe [s:b | gcb]       Probe I/O of block.
break <s:b | gcb>       Break before block is executed.
bafter <s:b | gcb>      Break after block is executed.
bshow s:b               Show block in system, s, with sorted list index, b.
clear <s:b | gcb>       Clear break point.
zcbreak                 Toggle: break at nonsampled zero crossing events?
zclist                  Display the nonsampled zero crossings list.
xbreak                  Toggle: break when step size is limited by a state?
tbreak [t]              Set/clear time break point.
nanbreak                Toggle: break on non-finite (NaN,Inf) values.
continue                Continue the simulation.
run                     Stop debugging and finish the simulation.
stop                    Stop execution.
quit                    Abort the simulation.
status [all]            Display debugging actions in effect.
states                  Display current state values.
systems                 Display a list of the model systems.
slist                   Display the sorted list(s).
minor                   Toggle: stop in minor time steps?
ishow                   Toggle: display integration information?
atrace level            Set algebraic loop trace level (0=none, 4=everything).
ashow <gcb | s:b>       Show algebraic loop involving block.
ashow s#n               Show algebraic loop, n, in system, s.
ashow clear             Remove any algebraic loop colorings.
<return>                Repeats last action.
Time is displayed as:
```

Ti=<time while in MinorTimeStep>

Tm=<time while in MajorTimeStep>

Ts=<integration start time>

(sldebug @0:0 'example53/反馈'):

8.3.10　ishow 命令

功能:允许或禁止显示积分信息.

语法:ishow

说明:

该命令在仿真期间,切 换显示积分信息.

例 8.20　在例 8.18 运行基础上,两次输入 ishow 命令.

```
sim('example53',[0,10],simset('debug','on'))
[Tm=0                         ] * * Start * * of system 'example53' outputs
(sldebug @0:0 'example53/反馈'): ishow
Display of integration information is enabled.
(sldebug @0:0 'example53/反馈'): ishow
Display of integration information is disabled.
(sldebug @0:0 'example53/反馈'):
```

参见:atrace 命令

8.3.11　minor 命令

功能:切换最小步模式.

语法:minor

说明:

该命令使得调试器进入或退出最小步模式.

参见:step 命令

8.3.12　nanbreak 命令

功能:设置或清除非限定值中断模式.

语法:nanbreak

说明:

nanbreak 命令使得仿真一遇到非限定值(NaN 或 Inf)时,就中断.如果已设置了非限定值中断模式,再运行该值将清除该模式.

参见:break,bafter,xbreak,tbreak,zcbreak 命令

8.3.13　next 命令

功能:进入下一个时间步开始.

语法:next

说明:

该命令计算在当前步剩下将要计算的模块，停在下一时间步的开始. 在执行 next 命令后，突出显示在下一时间步将要计算的第一个模块，并显示下一时间步时间.

参见：step 命令

8.3.14 probe 命令

功能：显示一个模块的输入和输出.

语法：probe [<s:b | gcb>]

说明：

probe 命令使得调试器进入或退出探测模式. 在探测模式下，调试器显示指定模块的输入/输出.

参见：disp、trace 命令

8.3.15 quit 命令

功能：中止仿真.

语法：quit

说明：

该命令中止当前仿真.

参见：stop 命令

8.3.16 run 命令

功能：运行仿真直到完成.

语法：run

说明：

run 命令从当前中断点运行仿真直到最终时间步. 它忽略中断点和显示点.

参见：continue，stop，quit 命令.

8.3.17 slist 命令

功能：列出模型的非虚拟模块.

语法：slist

说明：

slist 命令列出初调试模型的非虚拟模块. 列出模块索引和名字.

例 8.21 仍以 example53 为例.

```
sim('example53',[0,10],simset('debug','on'))
[Tm=0                    ] * * Start * * of system 'example53' outputs
(sldebug @0:0 'example53/反馈'): slist
---- Sorted list for 'example53' [6 blocks, 5 nonvirtual blocks, directFeed=1]
   0:0    'example53/反馈' (TransferFcn)
   0:1    'example53/Sum' (Sum)
   0:2    'example53/Out1' (Outport)
```

```
0:3      'example53/Transfer Fcn' (TransferFcn)
0:4      'example53/Out2' (Outport)
(sldebug @0:0 'example53/反馈'):
```

参见:systems

8.3.18 states 命令

功能:显示当前状态值.

语法:states

说明:

该命令显示模型当前状态列表.显示值、索引和每个状态名字列表.

例 8.22 仍以 example53 为例.

```
(sldebug @0:0 'example53/反馈'): states
Continuous state vector (value,index,name):
  0                          0 (0:0 'example53/反馈')
  0                          1 (0:3 'example53/Transfer Fcn')
  0                          2 (0:3 'example53/Transfer Fcn')
(sldebug @0:0 'example53/反馈'):
```

8.3.19 status 命令

功能:显示有效的调试选项.

语法:status

说明:

status 命令显示有效调试选项列表.

例 8.23 以 sample16 为例.

```
sim('sample16',[0,10],simset('debug','on'))
[Tm=0                        ] * * Start * * of system 'sample16' outputs
(sldebug @0:0 'sample16/Integrator'): status
  Current simulation time: 0 (MajorTimeStep)
  Default command to execute on return/enter: ""
  Stop in minor times steps: disabled
  Break at zero crossing events: disabled
  Break when step size is limiting by a state : disabled
  Time break point: disabled
  Break on non-finite (NaN,Inf) values: disabled
  Number of installed break points: 0
  Number of installed display points: 0
  Number of installed trace points: 0
  Display of integration information : disabled
  Algebraic loop tracing level: 0
```

(sldebug @0:0 'sample16/Integrator'):

8.3.20 step 命令

功能:步入下一模块.

语法:step

说明:

该命令在当前时间步评估下一个被计算的模块.在执行 step 命令后,调试器突出显示下一个要计算的模块和其输出连线.同时在调试器命令行中显示下一模块的名字.

参见:next 命令

8.3.21 stop 命令

功能:停止仿真.

语法:stop

说明:

stop 命令停止仿真.并从调试命令行方式,退至 MATLAB 命令行方式.

参见:continue,run,quit 命令

8.3.22 systems 命令

功能:列一个模型的非虚拟系统.

语法:systems

说明:

该命令在 MATLAB 命令行窗口中列一个模型的非虚拟系统.

例 8.24 以 FCsample 模型为例.比较 systems 和 slist 命令

```
sldebug('FCsample')
[Tm=0                          ] * * Start * * of system 'FCsample' outputs
(sldebug @0:0 'FCsample/Ramp/Clock'): systems
0     'FCsample'
(sldebug @0:0 'FCsample/Ramp/Clock'): slist
- - -     Sorted list for 'FCsample' [13 blocks, 11 nonvirtual blocks, directFeed=0]
  0:0     'FCsample/Ramp/Clock' (Clock, tid=0)
  0:1     'FCsample/Ramp/Constant' (Constant, tid=1)
  0:2     'FCsample/Constant' (Constant, tid=1)
  0:3     'FCsample/Ramp/Constant1' (Constant, tid=1)
  0:4     'FCsample/Ramp/Step' (Step, tid=0)
  0:5     'FCsample/Ramp/Sum' (Sum, tid=0)
  0:6     'FCsample/Ramp/Product' (Product, tid=0)
  0:7     'FCsample/Ramp/Sum1' (Sum, tid=0)
  0:8     'FCsample/Gain' (Gain, tid=0)
  0:9     'FCsample/Sum' (Sum, tid=0)
```

0:10 'FCsample/Scope' (Scope, tid=0)
(sldebug @0:0 'FCsample/Ramp/Clock'):

参见:slist 命令

8.3.23 tbreak 命令

功能:设置或清除时间断点.

语法:tbreak

tbreak t

说明:

该命令在指定的时间步设置断点.如果在指定的时间断点已经存在,tbreak 将清除该断点.如果没有指定时间,tbreak 将切换当前时间步的断点.

参见:break,bafter,xbreak,nanbreak,zcbreak 命令

8.3.24 trace 命令

功能:显示每次模块执行的输入和输出.

语法:trace gcb

trace s:b

说明:

该命令注册一个指定模块作为跟踪点,调试器显示每个注册模块每次执行的输入和输出.指定模块的参数 gcb 和 s:b 与前面一致.

参见:disp,probe,untrace,slist 命令

8.3.25 undisp 命令

功能:从调试器显示点列表中清除一个模块.

语法:undisp gcb

undisp s:b

说明:

该命令从调试器显示点列表中清除一个指定模块.指定模块的参数 gcb 和 s:b 与前面一致.

参见:disp,slist 命令

8.3.26 untrace 命令

功能:从调试器跟踪点列表中去除一个模块.

语法:untrace gcb

untrace s:b

说明:

untrace 命令从调试器跟踪点列表中去除一个指定模块.指定模块的参数 gcb 和 s:b 与前面一致.

参见:trace,slist 命令

8.3.27 xbreak 命令

功能:当调试器遇到步长极限状态时中断.

语法:xbreak

说明:

当调试器遇到求解器步长极限状态时,该命令中止模型的执行.如果 xbreak 模式已打开,运行 xbreak 命令,则关闭该模式.

参见:break,bafter,tbreak,nanbreak,zcbreak 命令

8.3.28 zcbreak 命令

功能:切换非采样过零事件中断.

语法:zcbreak

说明:

当非采样过零事件发生时,该命令使调试器中断.如果该模式已开,再运行 zcbreak 命令,则关闭该模式.

参见:break,bafter,xbreak,nanbreak,tbreak,zclist 命令

8.3.29 zclist 命令

功能:列出包含非采样过零点的模块.

语法:zchist

说明:

该命令列出能发生非采样过零点的模块.在 MATLAB 命令窗口中显示列表.

例 8.25 以 FCsample 模型为例.使用 zclist 命令

```
sldebug('FCsample')
[Tm=0                              ] **Start** of system 'FCsample' outputs
(sldebug @0:0 'FCsample/Ramp/Clock'): zclist
  0:4     'FCsample/Ramp/Step' (Step)
(sldebug @0:0 'FCsample/Ramp/Clock'):
```

参见:zcbreak 命令

第九章 Simulink 扩展工具 S 函数

9.1 S 函数概述

S 函数(系统函数)为参数化和扩展 Simulink 的能力提供了一种强大的机制.这里先介绍一下什么是 S 函数,在什么时候可能需要用到 S 函数,为什么要用到它.

9.1.1 什么是 S 函数

S 函数是对一个动态系统的计算机程序语言描述.S 函数可以使用 MATLAB 或者 C 语言写成.用 C 语言写成的 S 函数需要用 mex 工具编译成 MEX 文件.与其它的 MEX 文件一样,它们在需要的时候动态地链接进 MATLAB.

S 函数使用一种特殊的调用语法,通过它可以与 ODE 求解器进行交互.这种交互同求解器与 Simulink 内建模块之间的交互非常相似.

S 函数的形式非常全,它包括连续、离散和混合系统.因此,几乎所有的 Simulink 模型都可以描述为 S 函数.

通过 Nonlinear 库中的 S-Function 模块可以将 S 函数加进 Simulink 模型.使用 S-Function 模块对话框可以指定 S 函数的名字,如图 9.1 所示.

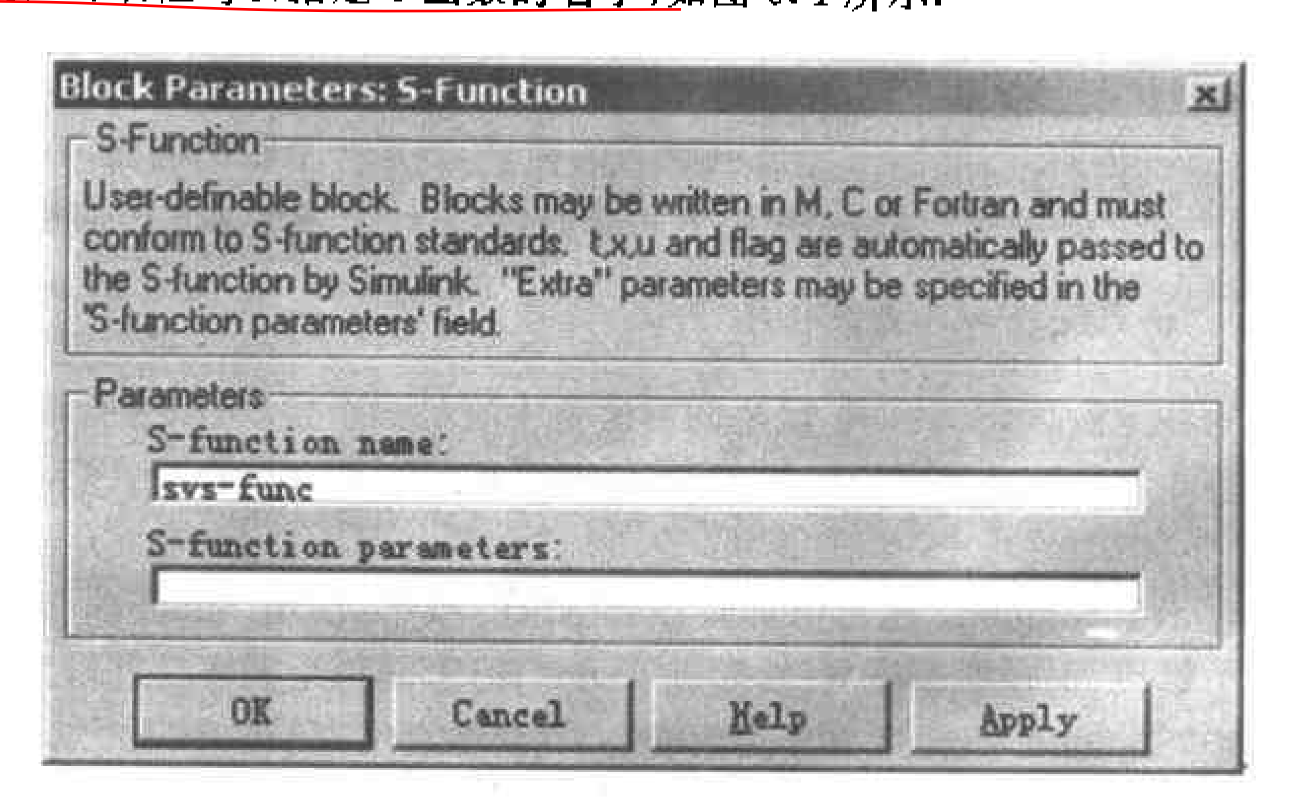


图 9.1 S 函数模块对话框

在该例中,模型包含有一个 S-Function 模块,模块引用的源程序的名字是:sys-func,它可以是一个 C MEX 文件或者是一个 M 文件.如果存在具有相同名字的 C MEX 文件和 M 文件,S 函数优先使用 C MEX 文件.

可以使用 Simulink 的模板工具为 S-Function 模块创建一个定制的对话框和图标.模

板对话框使得为 S 函数指定附加的参数变得更容易一些.

9.1.2 S 函数的作用与原理

S 函数最通常的用法是创建一个定制的 Simulink 模块. 可以在许多应用程序中使用 S 函数,包括:

1) 在 Simulink 中加进新的通用模块;

2) 将已存在的 C 代码合并入一个仿真中;

3) 将一个系统描述为一系列的数学方程;

4) 使用图形动画.

使用 S 函数的一个优点是可以创建一个通用的模块,在模型中可以多次使用它,使用时只需要改变它的参数值即可.

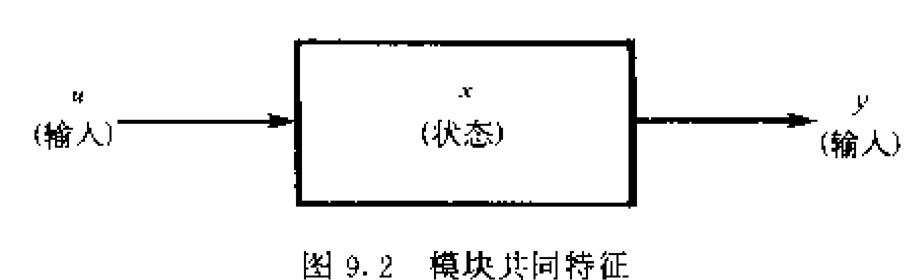


图 9.2 模块共同特征

Simulink 模型中的每一个模块都有如下的共同特征:一个输入向量 u,一个输出向量 y,和一个状态向量 x,如图 9.2 所示.

状态向量可能包括连续状态、离散状态或连续状态与离散状态的组合. 输入、输出和状态之间的数学关系可以用如式 9.1 所示的方程来表达:

$$\begin{aligned} y &= f_0(t,x,u) \quad (输出) \\ x_c' &= f_d(t,x,u) \quad (导数) \\ x_{d_{k+1}} &= f_u(t,x,u) \quad (更新) \end{aligned} \tag{9.1}$$

其中,$x = x_c + x_d$.

Simulink 将状态向量分为两部分:连续状态和离散状态. 连续状态占据着向量的第一部分,离散状态占据第二部分. 对于没有状态的模块,x 是一个空的向量. 在 MEX 文件 S 函数中,有两个独立的状态向量,一是离散状态向量,另一个是连续状态向量.

9.1.2.1 仿真阶段和 S 函数程序

在仿真的特定阶段,Simulink 反复调用模型中的每一个模块,以执行诸如计算输出、更新离散状态或者计算导数这样的任务. 其它调用是在仿真的开始或结束,以执行初始化和结束任务.

图 9.3 显示了 Simulink 执行仿真时各个阶段的顺序. 首先,Simulink 初始化模型;包括初始化每个模块,也包括 S 函数. 然后进入仿真循环,其每个循环经过作为一个仿真步. 在每个仿真步,Simulink 执行用户 S 函数模块,直到仿真结束.

Simulink 在模型中反复地调用 S 函数程序,以执行每一个阶段需要的任务. 这些任务包括:

1) 初始化. 在第一仿真循环之前,Simulink 初始化 S 函数. 此阶段 Simulink 执行以下工作:

初始化 SimStructure,SimStructure 是一个包含关于 S 函数信息的 Simulink 结构.

设置输入和输出端口的数量和大小.

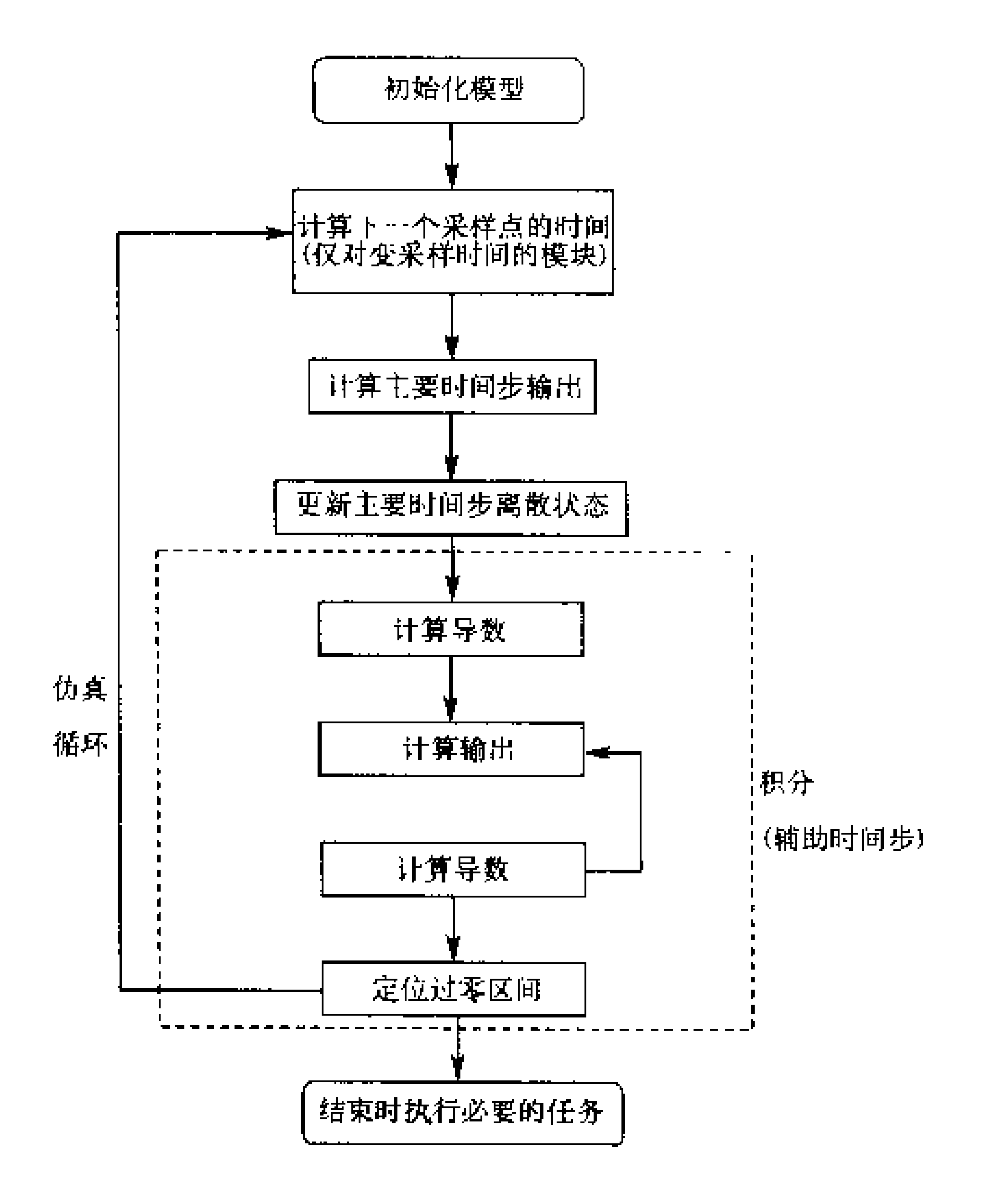


图 9.3 Simulink 执行仿真时各个阶段的顺序

设置模块采样时间.

分配存储空间并估计数组大小.

2) 计算下一采样点. 如果选择了变步长积分程序,该阶段计算下一变点的时间,即计算下一时间步.

3) 计算主要时间步的输出. 该调用完成后,模块的所有输出端口对于当前时间步是合法的.

4) 更新主要时间步的离散状态. 在该调用中,所有模块应当执行每时间步一次.

5) 积分. 这用于具有连续状态或非采样过零区间的模型. 如果 S 函数具有连续状态,Simulink 在辅助时间步调用 S 函数的输出和导数. 这就是 Simulink 可以计算 S 函数状态的原因. 如果 S 函数(仅为 C MEX)具有非采样过零区间,Simulink 将在辅助时间步调用 S 函数输出和过零成分,所以可以定位过零区间.

9.1.2.2 M 文件 S 函数和 C MEX 文件 S 函数

在 M 文件形式的 S 函数中,S 函数程序是作为 M 文件子程序的形式实现的. 在 C MEX 文件形式的 S 函数中,它们是以 C 语言函数实现的. S 函数程序的名字和它们执行的功能在 M 文件和在 C MEX 文件 S 函数中是相同的.

对于 M 文件形式的 S 函数,Simulink 传递一个标志参数给 S 函数,该标志表示当前的仿真阶段. 必须编写 M 代码调用合适的函数以得到每一个标志的值. 对于一个 C MEX

文件形式的 S 函数，Simulink 直接调用 S 函数程序. 表 9.1 列出了各个仿真阶段、相应的 S 函数程序和相关的 M 文件形式的 S 函数的标志值.

表 9.1　仿真阶段

仿真阶段	S 函数程序	标志(M 文件形式的 S 函数)
初始化	mdlInitializeSizes	flag=0
计算下一个采样点(可选)	mdlGetTimeOfNextVarHit	flag=4
计算输出	mdlOutputs	flag=3
更新离散状态	mdlUpdate	flag=2
计算导数	mdlDerivatives	flag=1
仿真任务结束	mdlTerminate	flag=9

C MEX 文件形式的 S 函数程序必须具有与表 9.1 列出的完全相同的名字. 在 M 文件形式的 S 函数中，必须提供基于标志值的调用合适的 S 函数程序的代码. 一个模板 M 文件 S 函数 sfuntmpl.m 位于/toolbox/simulink/blocks 目录中. 该模板使用一个 switch 语句处理标志的值. 所有需要做的就是将你的代码放在恰当的 S 函数程序中.

在 C MEX 文件形式的 S 函数中，Simulink 直接调用当前仿真阶段的适当的 S 函数程序. 一个用 C 语言写成的样板 S 函数 sfuntmpl.c 位于 simulink/src 目录中. 在同一目录下的 sfuntmpl.doc 是一个更为详细地带有注释的模板.

在开发 S 函数时，建议使用 M 文件或 C MEX 文件模板.

9.1.3　S 函数的有关概念

理解直接馈通、动态可变输入、设置采样时间和偏移量等这些关键的概念将有助于正确地创建 S 函数.

9.1.3.1　直接馈通

直接馈通意思是输出或可变采样时间直接由某一端口的输入值控制. 通常只要 S 函数满足如下条件，它就具有直接馈通：

1) 它的输出函数(mdlOutputs 或 flag==3)是输入的函数. 即在 mdlOutputs 函数中，如果使用了输入 u 的等式，就是直接馈通的. 输出可能还包括图形化输出，如有 XY Graph 显示器的情况下.

2) 它是一个可变采样时间的 S 函数(调用 mdlGetTimeOfNextVarHit 或 flag==4)，并且计算下一个采样点需要用到的输入 u.

正确地设置直接馈通非常重要，因为它会影响到模型中各个模块的执行顺序，并且它会被用来检测代数循环.

9.1.3.2　动态变化的输入

S 函数可以被编写成支持任意的输入宽度. 在这种情况，当仿真开始后，实际的输入个数是通过计算驱动 S 函数的输入向量的元素的个数来动态地确定的. 输入个数也可用来确定连续状态的个数，离散状态的个数和输出的个数.

要指明输入的个数是动态变化的，指定它的 sizes 结构的某些域的值为 -1，该结构是调用 mdlInitialize 返回的. 当调用 S 函数时，可以用 length(u)来确定实际的输入个数.

例如，图 9.4 显示了一个模型中的同一个 S 函数的两个实例.

图 9.4 S 函数示例

图 9.4 中上面的一个 S 函数模块由一个具有三个元素的输出向量的模块驱动，下面的一个 S 函数模块由一个具有一个标量输出的模块驱动. 通过指定该 S 函数模块具有动态变化的输入个数，可以使同一个 S 函数适用上面的两种情况. Simulink 自动地用具有合适的输入个数的向量调用该模块. 同样地，如果模块的其它诸如输出个数、离散状态或者连续状态这样的一些特征也可被指定为动态变化的，Simulink 确定这些向量的长度为与输入向量的长度相同.

9.1.3.3 设置各采样时间点及其偏移值

无论是 M 文件还是 C MEX 文件 S 函数，在指定何时执行 S 函数时，都具有较高的灵活性. Simulink 提供以下采样时间选择：

1) 连续采样时间；对于具有连续状态，和/或非采样过零区间的 S 函数. 这类 S 函数的输出在辅助时间步改变.

2) 连续但在辅助时间步采样时间固定；在每个主要仿真步需要执行，但在辅助时间步不改变值的 S 函数.

3) 离散采样时间；如果 S 函数模块的行为是离散时间间隔的一个函数，可以定义采样时间以控制什么时候 Simulink 调用该模块. 也可以定义每一个采样时间点的延迟偏移值. 偏移值的大小不能超过相应的采样时间.

一个采样时间点发生的时刻由下面的公式确定：

采样时间＝(n * 周期)＋偏移值

式中 n 是一个整数，它的第一个值是 0.

当采样时间给定后，Simulink 在每一个采样时间点调用 mdlOutput 和 mdlUpdate 程序，采样时间点由上面给出的公式确定. 在编写一个单一速率的离散 S 函数时，在输出和更新函数中没有必要明确地检测采样时间点. Simulink 只在适当的采样时间点上调用 S 函数.

4) 变采样时间；采样点间间隔可以变化的离散采样时间. 在每个仿真步开始，具有变采样时间的 S 函数为下一采样点时间积分.

5) 继承采样时间；有时候，一个 S 函数模块没有固有的采样时间属性(也就是说，它可能是连续的也可能是离散的，这取决于系统中另外一些模块的采样时间). 可以指定这些模块的采样时间为 inherited. 一个简单的例子是 Gain 模块从驱动它的模块那儿继承采样时间.

模块可以从驱动模块、目标模块、系统中最快采样时间中继承采样时间.

要设置一个模块的采样时间为从别处继承过来，将采样时间设为－1.

S 函数可以是单或多采样率，多采样率 S 函数具有多个采样时间.

采样时间以[sample_time,offset_time]这样的格式对来指定. 合法的采样时间对有：

[CONTINUOUS_SAMPLE_TIME, 0.0]

[CONTINUOUS_SAMPLE TIME, FIXED_IN MINOR_STEP_OFFSET]

[discrete_sample_time_period, offset]　　（需要实数值）

[VARIABLE_SAMPLE_TIME, 0.0]

其中：CONTINUOUS_SAMPLE_TIME = 0.0

FIXED_IN_MINOR_STEP_OFFSET = 1.0

VARIABLE_SAMPLE_TIME = −2.0

也可以指定采样时间由驱动模块继承．此时，只有一个采样时间对

[INHERITED_SAMPLE_TIME, 0.0]

或

[INHERITED SAMPLE_TIME, FIXED_IN_MINOR_STEP_OFFSET]

其中：INHERITED_SAMPLE_TIME = −1.0

指定采样时间时应遵循以下一些规则：

1）一个在辅助积分步改变的连续 S 函数，指定[CONTINUOUS_SAMPLE_TIME, 0.0]采样时间．

2)一个在辅助积分步不改变的连续 S 函数，应该指定：[CONTINUOUS_SAMPLE_TIME, FIXED_IN_MINOR_STEP_OFFSET]采样时间．

3）一个在指定速率下改变的离散 S 函数，应该指定离散采样时间对：[discrete_sample_time_period, offset]．其中 discrete_sample_period > 0.0，并且，0.0 ≤ offset < discrete_sample_period．

4）在变化速率下改变的离散 S 函数可以指定变步长离散采样时间：[VARIABLE_SAMPLE_TIME, 0.0]．

对于变步长离散任务，调用 mdlGetTimeOfNextVarHit 子程序来获取下一采样点时间．

如果 S 函数没有固有的采样时间，就必须继承采样时间．

5）一个即使在辅助积分步也随其输入改变而改变 S 函数，指定采样时间：[INHERITED_SAMPLE_TIME, 0.0]．

6）一个随输入改变而改变，但在辅助积分步不改变的 S 函数，就指定：[INHERITED SAMPLE_TIME, FIXED_IN_MINOR_STEP_OFFSET]采样时间．

9.1.4 S 函数的例子

有一些 S 函数的示例保存在 MATLAB 根目录下的如下两个子目录下：

simulink/src 目录下保存有 C MEX 文件；toolbox/simulink/blocks 目录下保存有 M 文件．

在 simulink/blocks 目录下包含有一些 M 文件形式的 S 函数．这些文件有：csfunc.m，dsfunc.m，vsfunc.m，mixed.m，vdpm.m，simom.m，simom2.m，limintm.m，sfun_varargm.m，vlimintm.m 和 vdlmintm.m．表 9.2 中列出了上这些 S 函数及其简要说明．

表 9.2 M 文件 S 函数例子

文件名	说明
csfunc. m	以状态空间形式定义一个连续系统
dsfunc. m	以状态空间形式定义一个离散系统
vsfunc. m	示范如何创建变步长模块. 该模块实现一个变步长延迟, 第一个输入延迟由第二个输入确定的时间
mixed. m	实现由连续一个积分器和一个单位延迟串联的混合系统
vdpm. m	实现 Van der Pol 等式
simom. m	一个具有 A, B, C, D 内部矩阵的状态空间 M 文件 S 函数例子. 该 S 函数实现: dx/dt=Ax+By; y=Cx+Du; 其中, x 是状态向量, u 是输入向量, y 是输出向量. A, B, C, D 矩阵嵌入 M 文件
simom2. m	一个具有 A, B, C, D 外部矩阵的状态空间 M 文件 S 函数例子. 状态空间结构与 simom. m 一样, 但 A, B, C, D 矩阵由文件参数外部提供
limintm. m	实现连续限定积分器, 其输出被限制在上下边界内, 并限制其初始条件
sfun_varargm. m	这是一个显示如何使用 MATLAB varargs 灵活性的 M 文件 S 函数例子
vlimintm. m	一个连续限定积分器 S 函数例子. 示范如何使用大小输入为 -1 来建立一个提供动态输入/状态宽度的 S 函数
vdlmintm. m	一个离散限定积分器 S 函数. 与 vlimintm. m 一样, 只是积分器是离散的

在 simulink/src 目录下还有 C MEX 文件 S 函数. 其中多数有 M 文件的 S 函数副本. 表 9.3 中列出这些 C MEX 文件 S 函数.

表 9.3 C MEX 文件 S 函数例子

文件名	说明
timestwo. c	一个将其输入乘以 2 的基本 C MEX 文件 S 函数
csfunc. c	定义一个连续系统的 C MEX 文件 S 函数例子
dsfunc. c	定义一个离散系统的 C MEX 文件 S 函数例子
dlimint. c	实现离散时间限定积分器
vsfunc. c	示范如何创建变步长模块. 该模块实现一个变步长延迟, 第一个输入延迟由第二个输入确定的时间
mixed. c	实现由连续一个积分器(1/s)和一个单位延迟(1/z)串联的混合动态系统
mixedmex. c	实现一个具有一个单输出和双输入的混合动态系统
quantize. c	向量化模块的 MEX 文件例子
resetint. c	复位积分器
sftable2. c	二维查找表 S 函数形式
sfun_dynsize. c	如何动态确定 S 函数输出大小的例子
sfun_errhndl. c	使用 mdlCheckParams S 函数子程序如何检查参数的例子
sfun_fcncall. c	一个配置函数调用子系统的 S 函数例子
sfun_multiport. c	具有多个输入输出端口的 S 函数例子
sfun_multirate. c	如何指定基于端口采样时间的 S 函数例子

续表 9.3

文件名	说明
sfun_zc.c	执行 abs(u)的具有非采样过零点的 S 函数例子.S 函数设计为变步长求解器
sfun_zc_sat.c	使用过零区间的饱和的例子
sfunmem.c	一个一积分步延迟和保持"存储"函数
vdpm.c	实现 Van der Pol 等式
simomex.c	实现一个单输出、双输入状态空间动态系统,其状态方程为:dx/dt=Ax+By;y=Cx+Du;其中,x 是状态向量,u 是输入向量,y 是输出向量.
stspace.c	实现一组状态空间方程,通过使用 S 函数模块和模板功能可以将该函数转变为一个新模块.该 MEX 文件例子与内置的状态空间(State-Space)模块功能一样.这是一个输入、输出、状态个数依赖工作空间传递参数的例子
stvctf.c	执行一个连续传递函数,传递函数多项式由输入向量传递.这对离散时间自适应控制应用非常有用
stvdct.f	实现离散传递函数,其传递函数多项式由输入向量传递.这对离散时间自适应控制应用非常有用
limintc.m	实现连续限定积分器
vdlmint.m	实现一个离散时间向量化限定积分器
vlimint.m	实现一个向量化限定积分器

9.2 编写 M 文件形式的 S 函数

定义 S 函数模块的 M 文件必须提供关于模型的一些信息,Simulink 在仿真的时候需要用到这些信息.在仿真期间,Simulink、ODE 求解器和 M 文件之间互相作用以执行具体的任务.这些任务包括定义初始状态和模块属性,计算导数、离散状态和输出.

Simulink 提供了一个 M 文件形式的 S 函数模板,它包括定义一些必要函数的语句和一些注释,这些注释有助于编写 S 函数模块中所必需的代码.模板文件 sfuntmpl.m 位于 MATLAB 根目录下的 toolbox/simulink/blocks 目录下.

M 文件 S 函数通过调用一系列 S 函数子程序来工作的,这些子程序是执行任务所需的 M 代码函数.表 9.4 列出了 M 文件 S 函数可用的 S 函数子程序.

创建 S 函数可以认为两个独立的任务:一个是初始化模块特性,包括输入、输出个数,连续和离散状态初始条件,采样时间;另一个是将算法放在合适的 S 函数子程序中.

表 9.4　M 文件 S 函数子程序

S 函数子程序	说　明
mdlInitializeSizes	定义基本 S 函数模块特性，包括采样时间，连续和离散状态的初始条件，数组大小
mdlDerivatives	计算连续状态变量导数
mdlUpdate	更新离散状态，采样时间，主要时间步
mdlOutputs	计算 S 函数的输出
mdlGetTimeOfNextVarHit	以绝对时间计算下一采样点的时间，该子程序仅用于在 mdlInitializeSizes 中指定一个可变离散采样时间的时候
mdlTerminate	执行仿真任务必要的结束

9.2.1　定义 S 函数模块的属性

要让 Simulink 能够识别一个 M 文件形式的 S 函数，必须提供该 S 函数的一些指定信息. 这些信息包括输入、输出和状态的个数以及模块的其它一些属性.

要提供这些信息给 Simulink，在 mdlInitializeSizes 的开始调用 simsizes 函数：

```
sizes=simsizes;
```

这一函数返回一个 sizes 结构. 表 9.5 列出了 sizes 结构的各个域并且说明了每个域中包含的信息：

表 9.5　sizes 结构

域　名	描　述
sizes. NumContStates	连续状态的个数
sizes. NumDiscStates	离散状态的个数
sizes. NumOutputs	输出的个数
sizes. NumInputs	输入的个数
sizes. DirFeedthrough	直接馈通标志
sizes. NumSampleTimes	采样时间的个数

在初始化好 sizes 结构之后，再调用一次 simsizes 函数：

```
sys=simsizes(sizes);
```

这一语句将 sizes 结构中的信息传递给 sys，以便将信息保存起来供 Simulink 使用.

9.2.2　M 文件形式的 S 函数的例子

下面给出一个简单的例子，例子中的 S 函数模块接受一个标量信号的输入并将它乘 2，如图 9.5 所示.

该 S 函数的 M 文件代码是基于 S 函数模板 sfuntmpl. m 生成的. 通过使用该模板，可以创建一个看起来与 C MEX 文件形式的 S 函数非常相似的 M 文件形式的 S 函数. 这就使得从 M 文件到 C MEX 文件的转换变得非常容易.

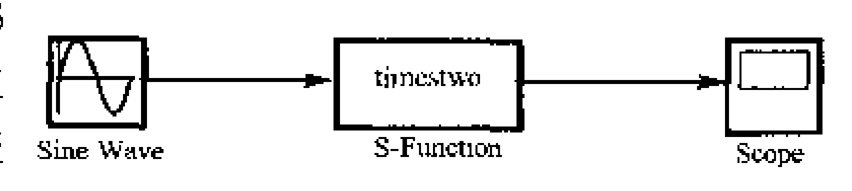


图 9.5　使用 S 函数模块的示例模型

下面是 timestwo. m 模块的 M 文件代码：

```
function [sys,x0,str,ts] = timestwo(t,x,u,flag)
% Dispatch the flag. The switch function controls the calls to
% S-function routines at each simulation stage.
switch flag,
    case 0
        [sys,x0,str,ts] = mdlInitializeSizes; % Initialization
    case 3
        sys = mdlOutputs(t,x,u); % Calculate outputs
    case{ 1, 2, 4, 9 }
        sys = []; % Unused flags
    otherwise
        error(['Unhandled flag = ',num2str(flag)]); % Error handling
% End of function timestwo
```

Simulink 传给 S 函数的头四个输入参数必须是变量 t，x，u 和 flag. t，时间；x，状态向量(即使没有状态变量，也必须提供)；u，输入向量；flag，控制在每一个仿真阶段调用哪个 S 函数子程序的参数.

Simulink 还要求输出参数 sys，x0，str 和 ts 按给定的顺序给出. 这些参数分别是：sys，是一个一般的返回参数，返回的值取决于 flag 的值，例如，如果 flag=3，sys 包含 S 函数的输出. x0，初始状态值(如果系统中没有状态将是一个空的向量). str，只是为了与 S 函数模块图的 API 一致而提供的，对于 M 文件形式的 S 函数，将它的值设为一个空矩阵. ts，一个包含有两列的矩阵，分别是与模块相关联的状态的采样时间及其偏移值；采样时间必须按递增的顺序声明，连续系统的采样时间设为 0.

下面是 timestwo. m 调用的 S 函数子程序：

```
%=====================================
% Function mdlInitializeSizes initializes the states, sample
% times, state ordering strings (str), and sizes structure.
%=====================================
function [sys,x0,str,ts] = mdlInitializeSizes
% Call function simsizes to create the sizes structure.
sizes = simsizes;
% Load the sizes structure with the initialization information.
sizes. NumContStates= 0;
sizes. NumDiscStates= 0;
sizes. NumOutputs= 1;
sizes. NumInputs= 1;
sizes. DirFeedthrough=0;
sizes. NumSampleTimes=1;
% Load the sys vector with the sizes information.
```

```
sys = simsizes(sizes);
%
x0 = []; % No continuous states
%
str = []; % No state ordering
%
ts = [-1 0]; % Inherited sample time
% end of mdlInitializeSizes
%==================================================
% Function mdlOutputs performs the calculations.
%==================================================
function sys = mdlOutputs(t,x,u)
sys = 2u;
% End of mdlOutputs.
```

要在 Simulink 中测试该 S 函数，在该 S 函数模块的输入连接一个正弦波生成器，将其输出连接到一个 Scope 模块. 现在可以运行仿真.

上面讨论的简单例子中没有状态变量. 大多数 S 函数模块需要对状态进行处理，不管它是连续的还是离散的. 下面讨论的系统的四种普通类型：连续、离散、混合和变步长都可以在 Simulink 中用 S 函数对它进行建模.

下面所有的例子都是基于 sfuntmpl. m 中的 M 文件形式的 S 函数模板.

9.2.2.1 连续状态的 S 函数的例子

Simulink 包含一个名为 csfunc. m 的函数，它是一个用 S 函数建模的连续状态系统的例子. 下面是该 M 文件形式的 S 函数的代码：

```
function [sys,x0,str,ts] = csfunc(t,x,u,flag)
% CSFUNC An example M-file S-function for defining a system of
% continuous state equations:
% x' = Ax + Bu
% y = Cx + Du
%
% Generate a continuous linear system:
A=[-0.09 -0.01
    1      0];
B=[ 1 -7
    0 -2];
C=[ 0  2
    1 -5];
D=[-3 0
    1 0];
```

```
%
% Dispatch the flag.
%
switch flag,
    case 0
        [sys,x0,str,ts]=mdlInitializeSizes(A,B,C,D); % Initialization
    case 1
        sys = mdlDerivatives(t,x,u,A,B,C,D); % Calculate derivatives
    case 3
        sys = mdlOutputs(t,x,u,A,B,C,D); % Calculate outputs
    case { 2, 4, 9 } % Unused flags
        sys = [];
    otherwise
        error(['Unhandled flag = ',num2str(flag)]); % Error handling
end
% end csfunc
%
%==============================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the
% S-function.
%==============================================================
%
function [sys,x0,str,ts] = mdlInitializeSizes(A,B,C,D)
%
% call simsizes for a sizes structure, fill it in and convert it
% to a sizes array.
%
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 2;
sizes.DirFeedthrough = 1; % Matrix D is nonempty.
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
%
% initialize the initial conditions
%
```

```
x0 = zeros(2,1);
%
% str is an empty matrix
%
str = [];
%
% Initialize the array of sample times; in this example the sample
% time is continuous, so set ts to 0 and its offset to 0.
%
ts = [0 0];
% end mdlInitializeSizes
%
%==============================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
%==============================================================
function sys = mdlDerivatives(t,x,u,A,B,C,D)
sys = A * x + B * u;
% end mdlDerivatives
%
%==============================================================
% mdlOutputs
% Return the block outputs.
%==============================================================
%
function sys = mdlOutputs(t,x,u,A,B,C,D)
sys = C * x + D * u;
% end mdlOutputs
```

与 timestwo.m 不同的是，当 flag=1 时，该例调用 mdlDerivatives 来计算连续状态变量的导数. 该系统的状态方程如下：

$$x' = Ax + Bu$$
$$y = Cx + Du$$

因此，一般的一系列连续微分方程都可用 csfunc.m 进行建模. csfunc.m 与内建的 State-Space 模块比较相似. 在对一个系数为时变的状态空间系统进行建模时，该 S 函数可以作为模板.

9.2.2.2 离散状态 S 函数的例子

Simulink 包括一个名叫 dsfunc.m 的函数，它是一个用 S 函数建模的离散状态系统的例子. 该函数与连续状态 S 函数的例子 csfunc.m 比较相似，惟一的不同是该例调用的是

mdlUpdate 而不是 mdlDerivative. 当 flag=2 时，mdlUpdate 更新离散状态. 对于一个单一采样率的离散 S 函数，Simulink 只在采样点上调用程序 mdlUpdate，mdlOutput 和 mdlGetTimeOfnextVarHit（如果需要）. 下面是 M 文件形式 S 函数的代码：

```
function [sys,x0,str,ts] = dsfunc(t,x,u,flag)
% An example M-file S-function for defining a discrete system.
% This S-function implements discrete equations in this form:
% x(n+1) = Ax(n) + Bu(n)
% y(n) = Cx(n) + Du(n)
%
% Generate a discrete linear system:
A=[-1.3839 -0.5097
     1.0000   0];
B=[-2.5559   0
     0      4.2382];
C=[ 0   2.0761
     0   7.7891];
D=[ -0.8141 -2.9334
     1.2426   0];
switch flag
    case 0
        sys = mdlInitializeSizes(A,B,C,D); % Initialization
    case 2
        sys = mdlUpdate(t,x,u,A,B,C,D); % Update discrete states
    case 3
        sys = mdlOutputs(t,x,u,A,B,C,D); % Calculate outputs
    case {1, 4, 9} % Unused flags
        sys = [];
    otherwise
        error(['unhandled flag = ',num2str(flag)]); % Error handling
end
% end of dsfunc
%==================================================
% Initialization
%==================================================
function [sys,x0,str,ts] = mdlInitializeSizes(A,B,C,D)
% call simsizes for a sizes structure, fill it in, and convert it
% to a sizes array.
sizes = simsizes;
sizes.NumContStates = 0;
```

```
sizes.NumDiscStates = 2;
sizes.NumOutputs = 2;
sizes.NumInputs = 2;
sizes.DirFeedthrough = 1; % Matrix D is non-empty.
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = ones(2,1); % Initialize the discrete states.
str = []; % Set str to an empty matrix.
ts = [1 0]; % sample time: [period, offset]
% end of mdlInitializeSizes
%==============================================================
% Update the discrete states
%==============================================================
function sys = mdlUpdates(t,x,u,A,B,C,D)
sys = A * x + B * u;
% end of mdlUpdate
%==============================================================
% Calculate outputs
%==============================================================
function sys = mdlOutputs(t,x,u,A,B,C,D)
sys = C * x + D * u;
% end of mdlOutputs
```

上面的例子符合本章前面讨论的仿真阶段. 系统的离散状态方程如下:

$$x(n+1)=Ax(n)+Bu(n)$$

$$y(n)=Cx(n)+Du(n)$$

因此,最普通的一系列差分方程都可用 dsfunc.m 进行建模. 它与内建的 Discrete State-Space 模块比较相似. 在给系数为时变的离散状态空间系统建模时,可用 dsfunc.m 作为模板.

9.2.2.3 混合系统 S 函数的例子

Simulink 包括一个名为 mixedm.m 的函数,它是一个用 S 函数进行建模的混合系统的例子(连续状态与离散状态的组合). 处理混合系统非常简单,参数 flag 使系统的连续部分和离散部分分别调用适当的 S 函数子程序. 混合 S 函数(或任何多采样率 S 函数)的一个奥妙是 Simulink 在所有的采样时间都调用程序 mdlUpdate, mdlOutput 和 mdlGetTimeOfNextVarHit, 这就意味着在这些程序中必须确定正在处理哪一个采样点,并且只对相应的采样点执行更新.

在 mixed.m 的模型中,有对一个连续 Integrator 后跟一个离散 Unit Delay. 用 Simulink 的模块图表示,如图 9.6 所示.

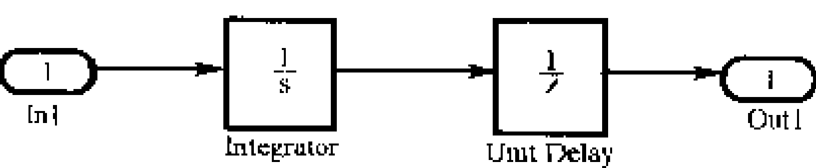


图 9.6 示例模型

下面是该 M 文件形式的 S 函数的代

码：

```
function [sys,x0,str,ts] = mixedm(t,x,u,flag)
% A hybrid system example that implements a hybrid system
% consisting of a continuous integrator (1/s) in series with a
% unit delay (1/z).
%
% Set the sampling period and offset for unit delay.
dperiod = 1;
doffset = 0;
switch flag
    case 0    % Initialization
     [sys,x0,str,ts] = mdlInitializeSizes(dperiod,doffset);
    case 1
        sys = mdlDerivatives(t,x,u); % Calculate derivatives
    case 2
        sys = mdlUpdate(t,x,u,dperiod,doffset); % Update disc states
    case 3
        sys = mdlOutputs(t,x,u,doffset,dperiod); % Calculate outputs
    case {4, 9}
        sys = []; % Unused flags
    otherwise
        error(['unhandled flag = ',num2str(flag)]); % Error handling
end
% end of mixedm
%
%==============================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the
% S-function.
%==============================================================
function [sys,x0,str,ts] = mdlInitializeSizes(dperiod,doffset)
sizes = simsizes;
sizes.NumContStates = 1;
sizes.NumDiscStates = 1;
sizes.NumOutputs = 1;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 2;
sys =simsizes(sizes);
```

```
x0  =ones(2,1);
str =[];
ts  =[0,  0      % sample time
      dperiod, doffset];
% end of mdlInitializeSizes
%
%=============================================================
% mdlDerivatives
% Compute derivatives for continuous states.
%=============================================================
%
function sys = mdlDerivatives(t,x,u,dperiod,doffset)
sys = u;
% end of mdlDerivatives
%
%=============================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time
% step requirements.
%=============================================================
%
function sys = mdlUpdate(t,x,u,dperiod,doffset)
% next discrete state is output of the integrator
% Return next discrete state if we have a sample hit within a
% tolerance of 1e-8. If we don't have a sample hit, return [] to
% indicate that the discrete state shouldn't change.
%
if abs(round((t-doffset)/dperiod)-(t-doffset)/dperiod) < 1e-8
      sys = x(1);
else
      sys = []; % This is not a sample hit, so return an empty
end % matrix to indicate that the states have not changed.
% end of mdlUpdate
%
%=============================================================
% mdlOutputs
% Return the output vector for the S-function
%=============================================================
%
```

```
function sys = mdlOutputs(t,x,u,doffset,dperiod)
% Return output of the unit delay if we have a
% sample hit within a tolerance of 1e-8. If we
% don't have a sample hit then return [] indicating
% that the output shouldn't change.
%
if abs(round((t-doffset)/dperiod)-(t-doffset)/dperiod) < 1e-8
        sys = x(2);
else
        sys = []; % This is not a sample hit, so return an empty
end % matrix to indicate that the output has not changed
% end of mdlOutputs
```

9.2.2.4. 变步长 S 函数的例子

名为 vsfunc.m 的 M 文件是使用变步长的一个 S 函数的例子. 该例当 flag=4 时调用 mdlGetTimeOfNextVarHit. 因为计算下一个采样时间取决于输入 u，所以该模块具有直接馈通. 通常，使用输入计算下一个采样时间(flag=4)的所有模块都需要直接馈通. 下面是 M 文件形式的 S 函数的代码：

```
function [sys,x0,str,ts] = vsfunc(t,x,u,flag)
% This example S-function illustrates how to create a variable
% step block in Simulink. This block implements a variable step
% delay in which the first input is delayed by an amount of time
% determined by the second input:
%
% dt = u(2)
% y(t+dt) = u(t)
%
switch flag,
    case 0
        [sys,x0,str,ts] = mdlInitializeSizes; % Initialization
    case 2
        sys = mdlUpdate(t,x,u); % Update Discrete states
    case 3
        sys = mdlOutputs(t,x,u); % Calculate outputs
    case 4
        sys = mdlGetTimeOfNextVarHit(t,x,u); % Get next sample time
    case { 1, 9 }
        sys = []; % Unused flags
    otherwise
```

```
        error(['Unhandled flag = ',num2str(flag)]); % Error handling
end
% end of vsfunc
%==============================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the
% S-function.
%==============================================================
%
function [sys,x0,str,ts] = mdlInitializeSizes
%
% call simsizes for a sizes structure, fill it in and convert it
% to a sizes array
%
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 1;
sizes.NumOutputs = 1;
sizes.NumInputs = 2;
sizes.DirFeedthrough = 1; % flag=4 requires direct feedthrough
% if input u is involved in
% calculating the next sample time
% hit.
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
%
% Initialize the initial conditions
%
x0 = [0];
%
% Set str to an empty matrix
%
str = [];
%
% Initialize the array of sample times
%
ts = [-2 0]; % variable sample time
% end of mdlInitializeSizes
%
```

```
%=============================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time
% step requirements.
%=============================================================================
%
function sys = mdlUpdate(t,x,u)
sys = u(1);
% end of mdlUpdate
%
%=============================================================================
% mdlOutputs
% Return the block outputs.
%=============================================================================
%
function sys = mdlOutputs(t,x,u)
sys = x(1);
% end mdlOutputs
%
%=============================================================================
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block. Note that the
% result is absolute time.
%=============================================================================
%
function sys = mdlGetTimeOfNextVarHit(t,x,u)
sys = t + u(2);
% end of mdlGetTimeOfNextVarHit
```

子程序 mdlGetTimeOfNextVarHit 返回"下一个采样点的时间",它是仿真过程中 vsfunc 下一次调用的时间. 这就意味着直到下一采样时间点之前,该 S 函数将不会有输出. 在 vsfunc 中,下一个时间点被设成 t+u(2),它意味着用第二个输入 u(2)来设置下一次调用 vsfunc 的时间.

9.2.2.5 传递附加参数

Simulink 经常传递 t,x,u 和 flag 给 S 函数. 传递附加参数给 M 文件形式的 S 函数是有可能的. limintm. m 就是这样的一个例子. 其代码如下:

```
function [sys,x0,str,ts]=limintm(t,x,u,flag,lb,ub,xi)
%LIMINTM Limited integrator implementation.
%    Example M-file S-function implementing a continuous limited integrator
```

```
%   where the output is bounded by lower bound (LB) and upper bound (UB)
%   with initial conditions (XI).
%   See sfuntmpl.m for a general S-function template.
%   See also SFUNTMPL.
%   Copyright (c) 1990  1998 by The MathWorks, Inc. All Rights Reserved.
%   $Revision: 1.13 $
switch flag
  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  case 0
    [sys,x0,str,ts] = mdlInitializeSizes(lb,ub,xi);
  %%%%%%%%%%%%%%%
  % Derivatives %
  %%%%%%%%%%%%%%%
  case 1
    sys = mdlDerivatives(t,x,u,lb,ub);
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % Update and Terminate %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  case {2,9}
    sys = []; % do nothing
  %%%%%%%%%%%%
  % Output %
  %%%%%%%%%%%%
  case 3
    sys = mdlOutputs(t,x,u);
  otherwise
    error(['unhandled flag = ',num2str(flag)]);
end
% end limintm
%
%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
%
function [sys,x0,str,ts] = mdlInitializeSizes(lb,ub,xi)
sizes = simsizes;
```

```
sizes.NumContStates   = 1;
sizes.NumDiscStates   = 0;
sizes.NumOutputs      = 1;
sizes.NumInputs       = 1;
sizes.DirFeedthrough  = 0;
sizes.NumSampleTimes  = 1;
sys = simsizes(sizes);
str = [];
x0  = xi;
ts  = [0 0];    % sample time: [period, offset]
% end mdlInitializeSizes
%==============================================================
% mdlDerivatives
% Compute derivatives for continuous states.
%==============================================================
function sys = mdlDerivatives(t,x,u,lb,ub)
if (x <= lb & u < 0)  | (x>= ub & u>0 )
  sys = 0;
else
  sys = u;
end
% end mdlDerivatives
%==============================================================
% mdlOutputs
% Return the output vector for the S-function
%==============================================================
function sys = mdlOutputs(t,x,u)
sys = x;
% end mdlOutputs
```

9.3 编写 C MEX 文件形式的 S 函数

9.3.1 C MEX 文件形式的 S 函数基本内容

一个定义 S 函数模块的 C MEX 文件也必须提供模型的有关信息；Simulink 在仿真的时候需要用到这些信息. 在仿真期间，Simulink、ODE 求解器和 MEX 文件之间互相作用以执行具体的任务. 这些任务包括定义初始状态和模块属性，计算导数、离散状态和各输出值.

C MEX 文件形式的 S 函数有着与 M 文件形式的 S 函数相同的结构并且执行相同的

功能. Simulink 包含一个编写 C MEX 文件形式的 S 函数的模板，名为 sfuntmpl. c.

C MEX 文件形式的 S 函数的一般格式如下：

```
#define S_FUNCTION_NAME your_sfunction_name_here
#define S_FUNCTION_LEVEL 2
#include "simstruc. h"
static void mdlInitializeSizes(SimStruct *S)
{
}
<additional S-function routines/code>
static void mdlTerminate(SimStruct *S)
{
}
#ifdef MATLAB_MEX_FILE /* Is this file being compiled as a
MEX-file? */
#include "simulink. c" /* MEX-file interface mechanism */
#else
#include "cg_sfun. h" /* Code generation registration
function */
#endif
```

mdlInitializeSizes 是 Simulink 与 S 函数交互时调用的第一个子程序. S 函数中还有一些其它 mdl * 子程序. 所有 S 函数必须依照这样的格式. 这就意味着不同的 S 函数可以用相同名字不同内容的 S 函数子程序来实现. 在调用 mdlInitializeSizes 子程序后，Simulink 再与其它各个子程序交互. 仿真最后，调用 mdlTerminate 子程序.

MEX 文件必须定义具体的函数，这些函数既提供了执行这些任务需要的信息，也包含了另外必需代码的一些语句. 表 9. 6 描述了 Simulink 在仿真期间调用的一些函数，它们是按调用的先后顺序给出的. 任一 S 函数 MEX 文件必须包含所有这些函数，即使一个模型并不一定要用到所有这些函数.

表 9. 6　在仿真时调用的函数

仿真阶段	S 函数子程序
对模块(输入输出向量的)大小的信息、采样时间和初始状态的初始化	mdlInitializeSizes mdlInitializeSampleTimes mdlInitializeConditions
计算各个输出	mdlOutputs
更新各个离散状态	mdlUpdate
计算下一个采样时间点(可选)	mdlGetTimeOfNextVarHit
计算各个连续状态的导数	mdlDerivatives
在仿真结束时执行的各项任务	mdlTerminate

与 M 文件形式的 S 函数不同的是，C MEX 文件形式的 S 函数没有与每一个 S 函数

子程序相联系的 flag 参数，这是因为 Simulink 在各个仿真阶段的合适的时候自动地调用各个 S 函数子程序。同样，C MEX 文件形式的 S 函数还有一些 S 函数子程序，而 M 文件形式的 S 函数没有与它们相对应的子程序。这些子程序包括 mdlInitializeSampleTimes 和 mdlInitializeConditions.

Simulink 用一个被称为 SimStruct 的数据结构维护 S 函数模块的有关信息。定义 SimStruct 的 include 文件 simstruc.h 提供了使得 MEX 文件可以从 SimStruct 中设置和取得值的宏.

Simulink 提供了一个 C MEX 文件形式的 S 函数的模板，里面包含有定义一些必要的功能的语句和一些注释，这些语句有助于编写 S 函数模块中所必需的代码。这一模板文件 sfuntmpl.c，可以在 MATLAB 根目录下的 Simulink/src 目录中找到。建议在开发 C MEX 文件形式的 S 函数时使用 C MEX 文件形式的模板.

(1) 在文件的开头所必需的语句

MEX 文件必须包含一条语句，该 include 语句定义了 SimStruct 数据结构，它保存有指向仿真时使用的数据的指针。该被包含进来的代码也定义了用来保存和获得 SimStruct 结构中的数据的宏。下面的这些语句必须在函数定义的前面：

```
#define S_FUNCTION_NAME your_sfunction_name_here
#define SFUNCTION_LEVEL 2
#include "simstruc.h"
```

将 your_sfunction_name_here 用 S 函数的名字代替就行了.

(2) 在文件的末尾所必需的语句

在 C MEX 文件形式的 S 函数的末尾必须包含下面的代码：

```
#ifdef MATLAB_MEX_FILE /?? Is this being compiled as MEX-file? ? /
#include "simulink.c" /?? MEX-file interface mechanism /
#else
#include "cg_sfun.h" /?? Code generation registration func /
#endif
```

这些语句为特定程序选择合适的代码.

如果文件被编译成 MEX 文件，simulink.c 将被包含进来。如果文件被用来与 Simulink Real-Time Workshop 链接以生成一个独立的或实时的可执行文件，cg_sfun.h 被包含进来.

(3) 条件编译 S 函数

S 函数可以在三种模式下进行编译.

MATLAB_MEX_FILE，表示 S 函数以 MEX 文件的形式创建.

RT 表示使用 Real-Time Workshop 为使用定步长求解器实时应用产生代码，来创建 S 函数.

NRT 表示使用 Real-Time Workshop 为使用变步长求解器非实时应用产生代码，来创建 S 函数.

(4) 错误处理

在使用 S 函数时，正确处理如不合法参数值等一些意外事件是非常重要的。如果 S 函

数需要检验其参数，使用下面的方法来报告错误：

```
ssSetErrorStatus(S,"error encountered due to ...");
return;
```

ssSetErrorStatus 的第二个参数必须驻留内存，不能是局部变量，如果使用下面的格式，将会出错.

```
mdlOutputs()
{
    char msg[256]; {ILLEGAL: to fix use "static char msg[256];"}
    sprintf(msg,"Error due to %s", string);
    ssSetErrorStatus(S,msg);
    return;
}
```

ssSetErrorStatus 错误处理方法，也可以使用 mexErrMsgTxt 函数，该函数使用例外处理来立即中止 S 函数执行，并返回 Simulink 控制. 为了在 S 函数中支持例外处理，Simulink 必须设置例外处理优先级.

没有例外的代码是没有条件转移的代码. 如果不调用 mexErrMsgTxt 函数或其它 API 子程序，使用 S 函数选项：SS_OPTION_EXCEPTION_FREE_CODE，在 mdlInitializeSizes 函数中使用下面的命令：

```
ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
```

设置这一选项，允许 Simulink 跳过通常在每个 S 函数调用前执行的例外处理设置，可以提高 S 函数的性能. 在使用 SS_OPTION_EXCEPTION_FREE_CODE 时，必须确保代码没有例外. 如果在该选项设置后 S 函数出现了意外情况，意想不到的结果就会发生.

所有 mex * 子程序都具有潜在的条件转移，另外如：mxGetPr，mxGetData，mxGetNumberOfDimensions，mxGetM，mxGetN，mxGetNumberOfElements，一些 mx * 子程序，也有潜在条件转移. 运行子程序可以去掉例外. 运行子程序有：mdlGetTimeOfNextVarHit，mdlOutputs，mdlUpdate，mdlDerivatives. 如果 S 函数中的运行子程序不包含例外，就可以设置无例外选项.

如果调用 ssSetErrorStatus 并从 S 函数返回时，Simulink 停止仿真并传递错误. 图 9.7 所示的流程图，确定了仿真是如何关闭的. 如果 ssSetErrorStatus 优先于 mdlStart 调用，不调用其它 S 函数子程序；如果 ssSetErrorStatus 在 mdlStart 中或后调用，将调用 mdlTerminate.

(5) 定义 S 函数模块的属性

SimStruct 中的 sizes 结构保存有关于 S 函数模块的必要的大小信息，包括输入、输出、状态的个数和另外的一些模块属性. 结构 sizes 在 mdlInitializeSizes 函数中初始化. 提供的各宏可用来设置结构中各个域的值. 如果某一个值没有被指定，它的值将被初始化为 0.

表 9.7 描述了 sizes 结构中的各个域，并说明了用来为各个域定义值的各个宏.

表 9.7 sizes 结构的各个域

结构的域	用来设置值的宏
连续状态的个数	ssSetNumContStates(S,numContStates)
离散状态的个数	ssSetNumDiscStates(S,numDiscStates)
输出的个数	ssSetNumOutputs(S,numOutputs)
输入的个数	ssSetNumInputs(S,numInputs)
直接馈通的标志	ssSetDirectFeedThrough(S,dirFeedThru)
采样时间的个数	ssSetNumSampleTimes(S,numSampleTimes)
输入参数的个数	ssSetNumSFcnParams(S,numInputArgs)
整数工作向量的元素的个数	ssSetNumIWork(S,numIWork)
实工作向量的元素的个数	ssSetNumRWork(S,numRWork)
指针工作向量的元素的个数	ssSetNumPWork(S,numPWork)

(6) 为 sizes 结构的各个域设置值

每一个宏的调用都有两个参数:第一个为 SimStruct 结构的句柄 S,第二个为数据的值.例如,下面的语句为一个 S 函数定义 sizes 结构:

```
static void mdlInitializeSizes(SimStruct S)
{
    ssSetNumContStates(S, 1);
    ssSetNumDiscStates(S, 0);
    ssSetNumOutputs(S, 1);
    ssSetNumInputs(S, 1);
    ssSetDirectFeedThrough(S, 0);
    ssSetNumSampleTimes(S, 1);
}
```

这一代码指明了该模块有一个连续状态并且没有离散的状态,有一个输出和一个输入,没有直接馈通,有一个采样时间(一个连续模块仍需定义一个值为 0 的采样时间).

9.3.2 S 函数子程序

图 9.7 显示了一个 C MEX 文件 S 函数子程序调用结构.包括可选的 S 函数子程序.图中粗线框中为必须调用子程序,细线框为可选调用子程序.

(1) Reat Time Workshop 交替调用结构

如果使用 Reat-Time Workshop 来为包含有 S 函数的模型产生代码,Simulink 并不完全调用图 9.7 所列的一系列过程.其调用过程如图 9.8 所示.图中粗线框中为必须调用子程序,细线框为可选调用子程序.

(2) 外部模式交替调用结构

当以外部模式运行 Simulink 时,调用 S 函数子程序的顺序要改变,图 9.9 给出了调用顺序.图中粗线框中为必须调用子程序,细线框为可选调用子程序.

(3) S 函数数据

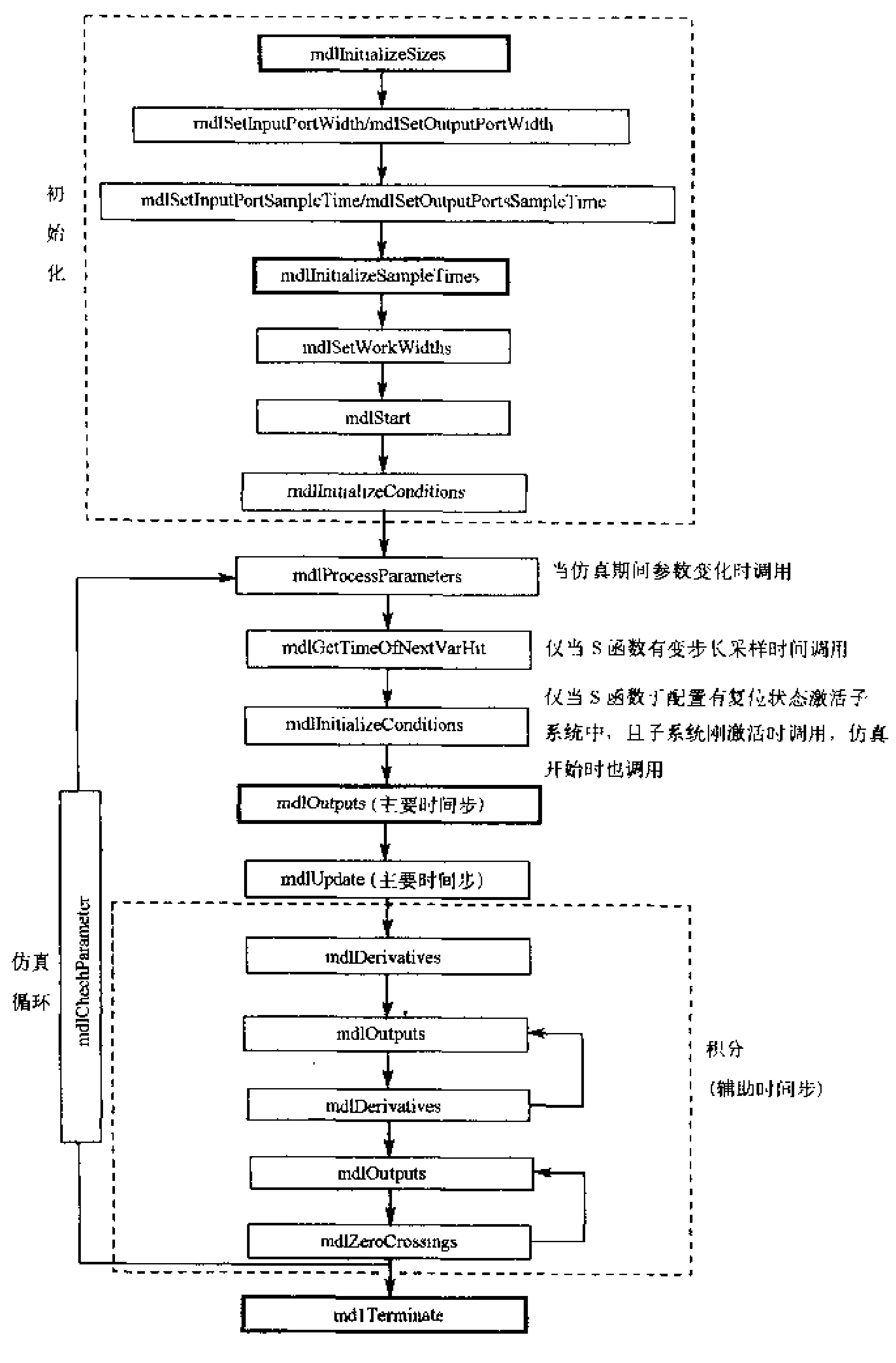


图 9.7 C MEX 文件 S 函数调用子程序流程

S 函数模块有输入和输出信号，内部状态以及其它通用工作区域. 一般讲，模块输入和输出是写入和读取自模块 I/O 向量. 模块输入也可以来自：由根输入端口模块的外部输入和接地. 模块输出也可以经由根输出端口模块到外部输出，另外输入和输出信号，S 函数具有：连续状态，离散状态，其它工作区域(如实数、整数或指针)工作向量.

S 函数模块可以通过使用 S 函数模块对话框传递参数给它们，对其参数化. 图 9.10

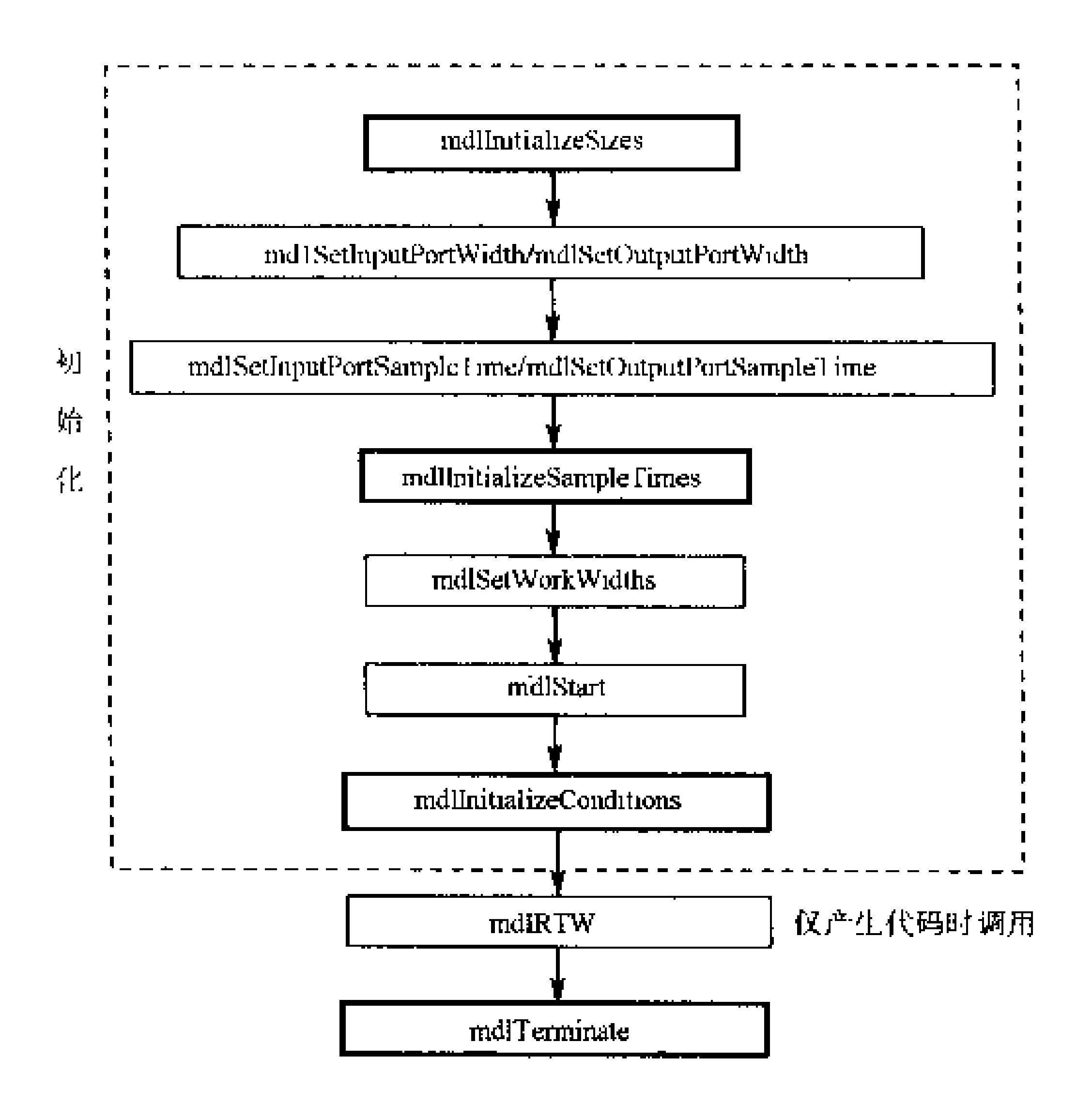


图 9.8　使用 Real Time Workshop 时 S 函数调用子程序流程

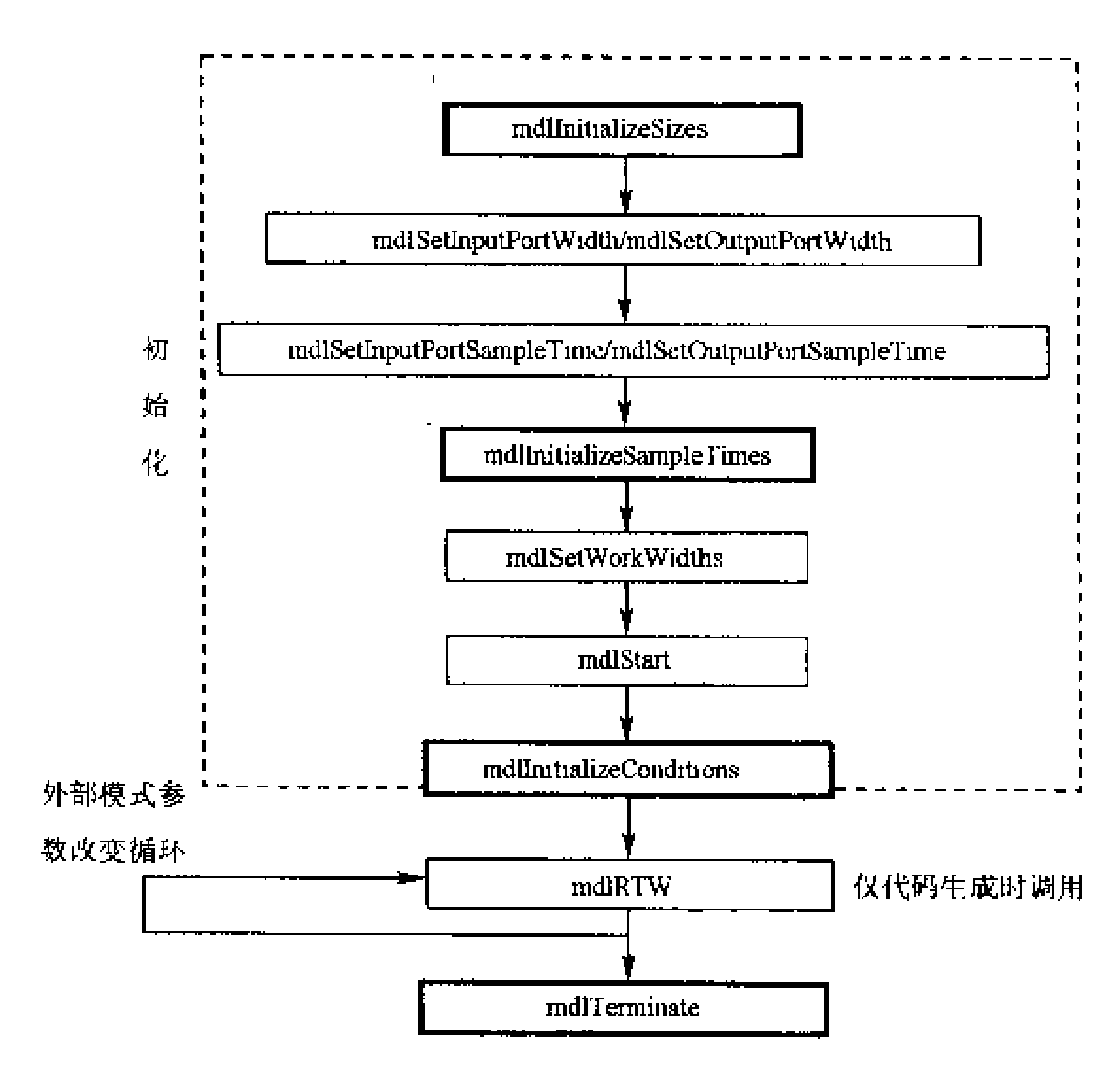


图 9.9　使用外部模式运行时 S 函数调用子程序流程

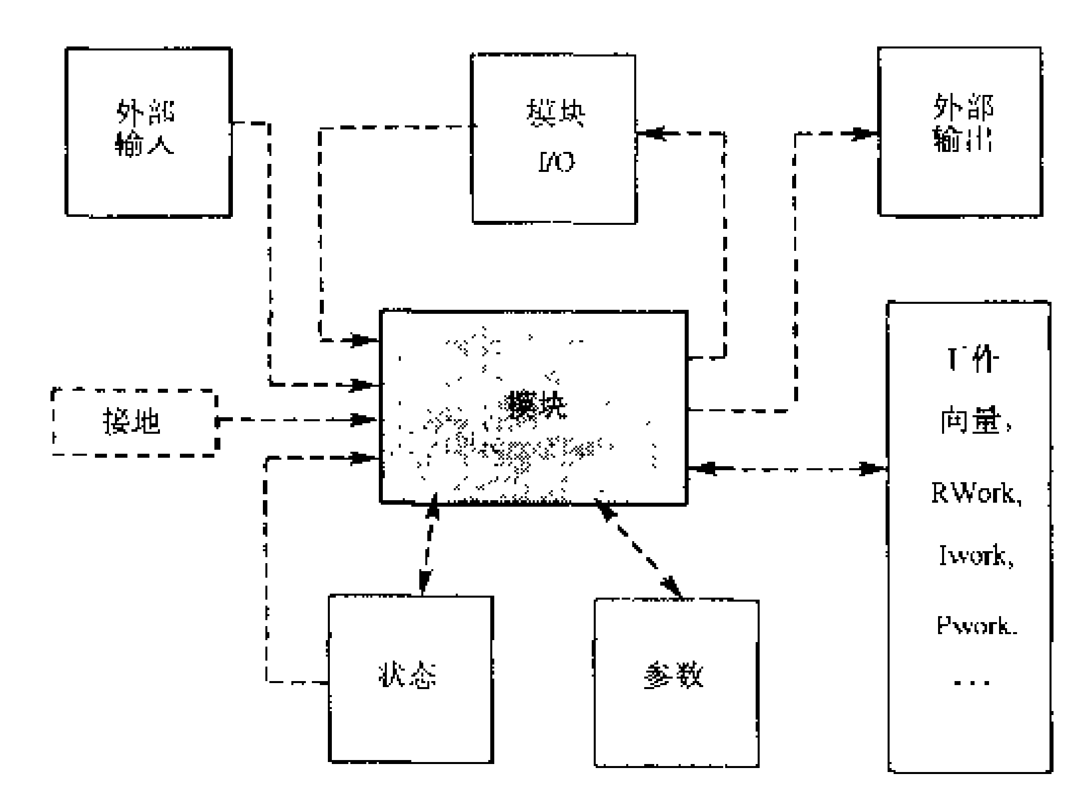


图 9.10 S 函数不同数据间的映射关系

显示了这些不同数据间的映射关系.

可变信号和向量的长度在 mdlInitializeSizes 子程序中配置,信号及其长度可以在仿真循环期间调用 S 函数子程序来访问. 在仿真循环期间访问输入信号,使用以下代码:

```
InputRealPtrs uPtrs = ssGetInputPortRealSignalPtrs(S,portIndex)
```

portIndex 从 0 开始,是一个指针数组,每个输入端口一个. 要访问这一信号的元素就使用:* uPtrs[element]

输入数组指针可能指向内存中不连续的地方,要获得输出信号可以使用代码:

```
real_T *y = ssGetOutputPortSignal(S,outputPortIndex);
```

为了正确在访问输入元素并将其写入输出元素中,使用下面代码:

```
int_T element;
int_T portWidth = ssGetInputPortWidth(S,inputPortIndex);
InputRealPtrs uPtrs = ssGetInputPortRealSignalPtrs(S,inputPortIndex);
real_T *y = ssGetOutputPortSignal(S,outputPortIdx);
for (elemet=0; element<portWidth; element++) {
  y[element] = *uPtrs[element];
}
```

通过指针算法来访问输入信号是错误的. 例如,在初始化 uPtrs 之后,使用:

```
real_T *u = *uPtrs; /* 不正确 */
```

面在上述内部循环中使用:

```
*y++ = *u++; /* 不正确 */
```

代码编译后,会导致 Simulink 崩溃,这是由于它可能访问不合法内存. 当不正确地访问输入信号时,如果输入到 S 函数的信号不是连贯的,就会发生崩溃. 当信号通过虚拟模块(Mux,Selector 等)时,就会发生信号不连贯. 要检验访问输入是否正确,传递一个复数信号给 S 函数的每个输入即可. 这可以通过创建一个 Mux 模块,使其输入端口数等于 S 函数信号所需宽度,将 Mux 的每个输入端口与驱动信号相连,如图 9.11 所示.

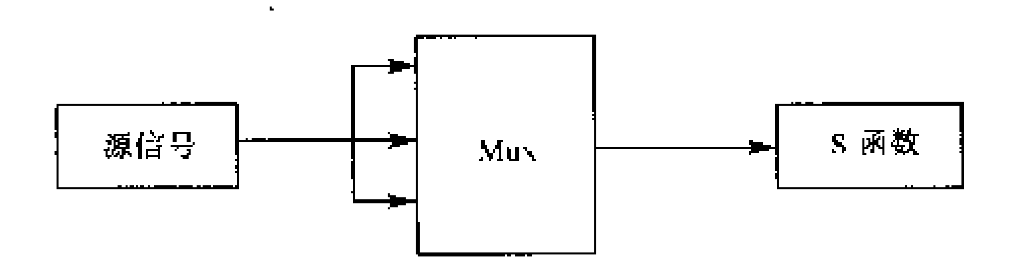


图 9.11　检验 S 函数访问内存的合法性

(4) 检查和处理 S 函数参数

除了其输入、输出和状态外，S 函数还可以接受参数. 这些参数可以用模块对话框中的 S-Function parameters 域来交互地设置. 如果交互地定义参数，在创建 S 函数时必须遵从如下步骤：

1) 确定将在模块对话框中指定参数的顺序.

2) 在 mdlInitializeSizes 函数中，使用 ssSetNumSFcnParams 宏来告诉 Simulink 有多少参数传给该 S 函数. 指定 S 作为第一个参数，并且指定交互定义的参数的个数作为第二个参数.

3) 使用宏 ssGetSFcnParam 获取 S 函数中的这些输入参数. 指定 S 作为第一个参数，在对话框中输入的参数的相对位置(0 为第一个位置)作为第二个参数.

在运行仿真时，在对话框的 S-Function parameters 域中指定参数的名字或值. 参数名字或值的顺序必须与前面的第一步中定义的顺序相同. 如果指定变量的名字，它们不必与 MEX 文件中使用的名字相同.

例如，下面的代码是一个设备驱动程序 S 函数的一部分. 使用的四个输入参数分别是：BASE_ ADDRESS，GAIN_ RANGE_ PRM，PROG_ GAIN_ PRM 和 NUM_ OF_ CHANNELS_PRM. 代码使用 #define 语句将特定的输入参数与参数的名字联系起来.

```
/* Input Parameters */
#define BASE_ADDRESS_PRM(S)    ssGetSFcnParam(S, 0)
#define GAIN_RANGE_PRM(S)    ssGetSFcnParam(S, 1)
#define PROG_GAIN_PRM(S)    ssGetSFcnParam(S, 2)
#define NUM_OF_CHANNELS_PRM(S)    ssGetSFcnParam(S, 3)
```

运行仿真时，用户在模块对话框的 S-Function parameters 域中输入四个变量名字或值. 第一个对应于第一个期望的参数 BASE_ADDRESS_PRM(S). 第二个相应于下一个期望的参数，如此等等.

函数 mdlInitializeSizes 包含有下面的语句：

```
ssSetNumSFcnParams(S, 4);
```

Simulink 经常要传递 t，x 和 u 给 S 函数. 这些参数没有包含在 S 函数参数中. 这里只包括需要传递给 S 函数的附加参数.

(5) 改变参数

要通知一个 S 函数的参数改变了，该 S 函数必须注册 mdlCheckParameters 子程序. 这一子程序将会在任何一次调用 mdlInitializeSizes 后被调用. 如果要处理改变的参数，注册 mdlCheckParameters 子程序. 该子程序将参数缓存在工作向量中. 在写含有仿真期间参数可变的 S 函数时，可选 S 函数子程序 mdlCheckParameters 和 mdlProcessParameters，

一般可同时使用.这些函数只有在 C MEX 文件 S 函数中可以使用.

S 函数包含这些子程序,在使用 Real-TimeWorkshop 工作时,必须检验其代码稳定处理参数改变.

1) mdlCheckParameters 子程序.使用 mdlCheckParameters 子程序来检验参数设置的正确性:

```
#define MDL_CHECK_PARAMETERS /* define is required for use */
#if defined (MDL_CHECK_PARAMETERS)
static void mdlCheckParameters(SimStruct *S)
{
}
#endif
```

仿真期间,无论何时参数改变或重新计算,该子程序都验证新参数.

在仿真运行时,S 函数参数改变可能随时发生,既可能在仿真的开始步,也可能在仿真中间步.如果在仿真中间步改变,Simulink 调用子程序两次,来处理参数改变.第一次调用是用来检验参数是否正确,检验新参数后,仿真继续使用原参数值,直到下一仿真步,此时才使用新参数值.需要冗余调用来维持仿真的稳定性.

在该子程序中不能访问工作空间、状态、输入、输出以及子程序中的其它向量.该子程序仅用来检验参数,另外的处理需要在 mdlProcessParameters 中来进行.

下面一段代码是检验 S 函数第一个参数是否为非负实标量.

```
#define PARAM1(S) ssGetSFcnParam(S,0)
#define MDL_CHECK_PARAMETERS /* 改变为 #undef to remove function */
#if defined(MDL_CHECK_PARAMETERS) && defined(MATLAB_MEX_FILE)
static void mdlCheckParameters(SimStruct *S)
{
  if (mxGetNumberOfElements(PARAM1(S)) != 1) {
  ssSetErrorStatus(S,"Parameter to S-function must be a scalar");
return;
} else if (mxGetPr(PARAM1(S))[0] < 0) {
  ssSetErrorStatus(S, "Parameter to S-function must be non-negative");
return;
  }
}
#endif /* MDL_CHECK_PARAMETERS */
```

因为 mdlProcessParameters 子程序只有在仿真运行时才调用,除了上面一段代码外,在初始化时,必须在 mdlInitializSizes 中加入一个对该子程序的调用来检验参数.为此,mdlInitializSizes 在使用 ssSetNumSFcnParams 设置完参数个数后,应使用下面代码:

```
static void mdlInitializeSizes(SimStruct *S)
{
  ssSetNumSFcnParams(S, 1); /* 预期参数个数 */
```

```
#if defined(MATLAB_MEX_FILE)
  if(ssGetNumSfcnParams(s) == ssGetSFcnParamsCount(s) {
  mdlCheckParameters(S);
  if(ssGetErrorStates(S) != NULL) return;
} else {
  return; /* Simulink 将报告不匹配错误. */
}
#endif
...
}
```

2) mdlProcessParameters 子程序. mdlProcessParameters 是一个可选执行子程序,它在 mdlCheckParameters 子程序改变和检验参数后执行. 它在仿真循环顶部执行,该子程序只有用于 C MEX 形式 S 函数中. 该子程序主要用于处理新改变的参数. 在使用 RTW 时,仿真不调用该子程序. 因此,在使用 RTW 时,必须编写不依赖该子程序的代码. 为此,要在 S 函数中嵌入一段目标语言编译器代码. 基本形式如下:

```
#define MDL_PROCESS_PARAMETERS /* 改变为#undef to remove function */
#ifdefined(MDL_PROCESS_PARAMETERS) && defined(MATLAB_MEX_FILE)
  static void mdlProcessParameters(SimStruct *S)
  {
  }
  #endif /* MDL_PROCESS_PARAMETERS */
```

下面是一段处理字符串参数的例子代码,mdlProcessParameters 检验"+-+"字符串形式.

```
#define MDL_PROCESS_PARAMETERS /* 改变为 #undef to remove function */
#if defined(MDL_PROCESS_PARAMETERS) && defined(MATLAB_MEX_FILE)
  static void mdlProcessParameters(SimStruct *S)
  {
  int_T i;
  char_T *plusMinusStr;
  int_T nInputPorts = ssGetNumInputPorts(S);
  int_T *iwork = ssGetIWork(S);
  if ((plusMinusStr=(char_T *)malloc(nInputPorts+1)) == NULL) {
    ssSetErrorStatus(S,"Memory allocation error in mdlStart");
    return;
  }
  if (mxGetString(SIGNS_PARAM(S),plusMinusStr,nInputPorts+1) != 0) {
    free(plusMinusStr);
    ssSetErrorStatus(S,"mxGetString error in mdlStart");
    return;
```

```
}
for (i = 0; i < nInputPorts; i++) {
  iwork[i] = plusMinusStr[i] == '+'? 1: -1;
}
free(plusMinusStr);
}
#endif /* MDL_PROCESS PARAMETERS */
```

mdlProcessParameters 在仿真循环开始之前，从 mdlStart 字符中调用，调入符号字符串. 代码为

```
#define MDL_START
#if defined(MDL_START)
static void mdlStart(SimStruct *S)
{
  mdlProcessParameters(S);
}
#endif /* MDL_START */
```

(6) 定义 S 函数模块特性

在 SimStruct 中结构 sizes 保存着有关 S 函数模块基本的大小信息. 包括输入、输出、状态个数以及其它模块特性. 结构 sizes 在 mdlInitializeSizes 函数中初始化. 提供宏为结构字段设置值. 不指定，则设为 0.

当与 S 函数交互时，mdlInitializeSizes 是 Simulink 调用的第一个子程序. 该子程序指定 S 函数 sizes 信息来确定模块的特性. 每个输入端口的直接馈通标志可设为 1 或 0. 如果输入 u 在 mdlOutput 或 mdlGetTimeOfNextVarHit 子程序中使用，则应设置为 1；将直接馈通标志设为 0，即告诉 Simulink 输入 u 将在这些 S 函数子程序中不使用. 不按这一准则，将导致不可预测的结果.

可以设置参数：NumContStates，NumDiscStates，NumInputs，NumOutputs，NumRWork，NumIWork，NumPWork，NumModes，和 NumNonsampledZCs 为定点非负整数，或者告诉 Simulink 动态调整其大小.

DYNAMICALLY_SIZED，设置状态、工作向量长度，从驱动模块继承. 除了用 mdlSetWorkWidths 设置宽度外，根据标量扩展规则，设置宽度为实际输入的宽度.

0 或负数，设置长度（或宽度）为指定值. 缺省为 0.

另外还可以使用宏来设置和检验 mdlInitializeSizes 子程序中的信息.

1）在设置输入、输出端口数后，使用 ssGetInputPortConnected(S) 和 ssGetOutputPortConnected(S)宏来确定端口是否连接.

2）当一个参数在仿真期间不能改变，则使用：ssSetSFcnParamNotTunable(S, paramIdx)，其中 paramIdx 从 0 开始. 当一个参数已经被指定为“不可调”，如果试图改变参数，Simulink 在仿真期间将出现一个错误.

3）如果 S 函数输出是离散的，指定：SS_OPTION_DISCRETE_VALUED_OUTPUT.

4) 可以使用 ssGetPath(Simulink 模块的全模型路径)和 ssGetModelName(S 函数名字),如果它们一样,即 strcmp(ssGetPath(S),ssGetModelName(S))==0,则 S 函数正从 MATLAB 命令行执行,不是仿真的一部分.

在 mdlInitializeSizes 子程序中也必须指定模块的采样时间数,指定采样时间两种方法:基于端口采样时间和基于模块采样时间.下面的代码,给出了该函数的基本形式.

```
/* Function: mdlInitializeSizes ======================= *
Abstract:
 * The sizes information is used by Simulink to determine the S-function
 * block's characteristics (number of inputs, outputs, states, etc.).
 *
 * The direct feedthrough flag can be either 1=yes or 0=no. It should be
 * set to 1 if the input, "u", is used in the mdlOutput function. Setting
 * this to 0 is tells Simulink that "u" will not be used in the
 * mdlOutput function. If you violate this, unpredictable results
 * will occur.
 *
 * The NumContStates, NumDiscStates, NumInputs, NumOutputs, NumRWork,
 * NumIWork, NumPWork NumModes, and NumNonsampledZCs widths can be set
to:
 * DYNAMICALLY_SIZED - In this case, they will be set to the actual
 * input width, unless you are have a
 * mdlSetWorkWidths to set the widths.
 * 0 or positive number - This explicitly sets item to the specified
 * value.
 */
static void mdlInitializeSizes(SimStruct *S)
{
ssSetNumSFcnParams(S, 0); /* Number of expected parameters */
if (ssGetNumSFcnParams(S) ! = ssGetSFcnParamsCount(S)) {
/*
 * If the the number of expected input parameters is not equal
 * to the number of parameters entered in the dialog box return.
 * Simulink will generate an error indicating that there is a
 * parameter mismatch.
 */
return;
}
ssSetNumContStates( S, 0); /* number of continuous states */
ssSetNumDiscStates( S, 0); /* number of discrete states */
```

```
/*
 * Configure the input ports. First set the number of input ports,
 * then set for each input port index starting at 0, the width
 * and wether or not the port has direct feedthrough (1=yes, 0=no).
 * The width of a port can be DYNAMICALLY_SIZED or greater than zero.
 * A port has direct feedthrough if the input is used in either
 * the mdlOutputs or mdlGetTimeOfNextVarHit functions.
 */
if (! ssSetNumInputPorts(S, nInputPorts)) return;
ssSetInputPortWidth(S, inputPortIdx, width);
ssSetInputPortDirectFeedThrough(S, inputPortIdx, needsInput);
/*
 * Configure the output ports. First set the number of output ports,
 * then set for each output port index starting at 0, the width
 * of the output port which can be DYNAMICALLY_SIZE or greater than zero.
 */
if (! ssSetNumOutputPorts(S, nOutputPorts)) return;
ssSetOutputPortWidth(S, outputPortIdx, width);
/*
 * Set the number of sample times. This must be a positive, nonzero
 * integer indicating the number of sample times or it can be
 * PORT_BASED_SAMPLE_TIMES. For multi-rate S-functions, the
 * suggested approach to setting sample times is via the port
 * based sample times routine. When you create a multirate
 * S-function, care needs to be taking to verify that when
 * slower tasks are preempted that your S-function correctly
 * manages data as to avoid race conditions. When port based
 * sample times are specified, the block cannot inherit a constant
 * sample time at any port.
 */
ssSetNumSampleTimes( S, 1); /* number of sample times */
/*
 * Set size of the work vectors.
 */
ssSetNumRWork( S, 0); /* number of real work vector elements */
ssSetNumIWork( S, 0); /* number of integer work vector elements */
ssSetNumPWork( S, 0); /* number of pointer work vector elements */
ssSetNumModes( S, 0); /* number of mode work vector elements */
ssSetNumNonsampledZCs( S, 0); /* number of nonsampled zero crossings */
```

```
/*
 * Set any S-function options which must be OR'd together.
 * The available options are:
 * SS_OPTION_EXCEPTION_FREE_CODE - if your S-function does not use
 * mexErrMsgTxt, mxCalloc, or any other routines which can throw an
 * exception when called, you can set this option for improved
 * performance.
 *
 * SS_OPTION_RUNTIME_EXCEPTION_FREE_CODE - Similar to
 * SS_OPTION_EXCEPTION_FREE_CODE except it only applies to the "run-time"
 * routines: mdlGetTimeOfNextVarHit, mdlOutputs, mdlUpdate, and
 * mdlDerivatives.
 *
 * SS_OPTION_DISCRETE_VALUED_OUTPUT - This should be specified if your
 * S-function has a discrete valued outputs. This is checked when
 * your S-function is placed within an algebraic loop. If your S function
 * has discrete valued outputs, then its outputs will not be assigned
 * algebraic variables.
 *
 * SS_OPTION_PLACE_ASAP - This is used to specify that your S-function
 * should be placed as soon as possible. This is typically used by
 * devices connecting to hardware.
 *
 * SS_OPTION_ALLOW_INPUT_SCALAR_EXPANSION - This is used to specify
 * that the input to your S-function input ports can be either 1 or
 * the size specified by the port which is usually referred to as
 * the block width.
 *
 * SS_OPTION_DISALLOW_CONSTANT_SAMPLE_TIME
 *                                  - This is used to disable
 * your S function block from inheriting a constant sample time.
 *
 * SS_OPTION_ASYNCHRONOUS - This option applies only to S-functions that
 * have no input ports and 1 output port. The output port must be
 * configured to perform function calls on every element. If any of
 * these requirements are not met, the SS_OPTION_ASYNCHRONOUS is
 * ignored. Use this option when driving function-call subsystems that
 * will be attached to interupt service routines.
 *
```

```
* SS OPTION_ASYNC_RATE_TRANSITION - Use this when your s-function converts
* a signal from one rate to another rate.
*
* SS_OPTION_RATE_TRANSITION - Use this why your S-function is behaving
* as a unit delay or ZOH. This option is only supported for these two
* operations. An unit delay operation is identified by the presence
* of mdlUpdate and if not present then the operation is ZOH.
*/
ssSetOptions( S, 0); /* general options (SS_OPTION_xx) */
} /* end mdlInitializeSizes */
```

如果开发定制多端口S函数模块，端口数依据参数而变化，并且要将其放入Simulink库中，那么就必须指定模块定制改变外观，为此，在模型保存之前，在MATLAB命令行中执行如下命令：

set_param('block','MaskSelfModifiable','on')

如果指定模块定制改变外观失败，意味着，当库连接时，其端口数与其库中显示的不匹配.

(7) 配置输入和输出端口特性

在mdlInitializeSizes中，可以指定端口宽度为正非零整数，仿真循环期间，当S函数子程序访问时，相应的输入/输出信号将具有指定的宽度. 然而，如果为一个输入端口指定：SS_OPTION_ALLOW_INPUT_SCALAR_EXPANSION和正非0整数N，那么，相应信号的宽度将为1或N. 在仿真循环期间，可以在子程序调用过程中使用ssGetInputPortWidth来确定实际使用的端口宽度.

还可以指定输入、输出信号动态可调，即由DYNAMICALLY_SIZED定义，当其在模型中传播向量宽度时，Simulink将确定相应的宽度. 可以通过：mdlSetInputPortWidth和mdlSetOutputPort子程序来影响其向量传播宽度. 如果S函数未指定这些子程序，则使用缺省标量扩展规则.

如果处理动态改变输入、输出大小不能满足需要，就必须使用可选执行子程序：mdlSetInputPortWidth和mdlSetOutputPortWidth.

在设置任何属性之前，必须正确指定端口数. 如果试图设置一个不存在的端口属性，就会访问不合法内存，从而导致Simulink崩溃.

1) mdlSetInputPortSampleTime子程序. 对于一个继承采样时间输入端口，该子程序调用时，使用假设的采样时间. 如果假设的采样时间可接受，子程序继续，并使用ssSetInputPortSampleTime设置实际端口采样时间；如果假设的采样时间不可接受，则通过ssSet ErrorStatus产生一个错误. 任何由给定端口采样时间明确定义采样时间的其它继承的输入或输出端口，也可以通过调用：ssSetInputPortSampleTime或ssSetOutputPortSampleTime来设置其宽度. 当基于采样时间的继承端口指定后，采样时间将为：连续，[0.0, 0.0]；或离散，[周期，偏移量]，其中周期大于0.0，小于无穷大，偏移量大于等于0.0、小于周期.

常数的、激活的和可变步长的采样时间，不会传播给具有基于采样时间端口的 S 函数.

MdlSetInputPortSampleTime 子程序其代码形式为

```
#if defined(MDL_SET_INPUT_PORT_SAMPLE_TIME)
                              && defined(MATLAB_MEX_FILE)
static void mdlSetInputPortSampleTime(SimStruct *S,
                                      int_T portIdx,
                                      real_T sampleTime,
                                      real_T offsetTime)
{
}
#endif /* MDL_SET_INPUT_PORT_SAMPLE_TIME */
```

2) mdlSetOutputPortSampleTime 子程序. 对于一个继承采样时间输出端口，该子程序调用时，使用假设的采样时间. 如果假设的采样时间可接受，子程序使用 ssSetOutputPortSampleTime 设置实际端口采样时间；如果假设的采样时间不可接受，则通过 ssSetErrorStatus 产生一个错误. 任何由给定端口采样时间明确定义采样时间的其它继承的输入或输出端口，也可以通过调用：ssSetInputPortSampleTime 或 ssSetOutputPortSampleTime 来设置其宽度. 通常情况下，采样时间是前向传播的，然而，如果馈源模块具有继承采样时间，就会选择反向传播来得到模块采样时间. 如果采用反向传播采样时间，Simulink 为所有继承输出端口信号连续调用该子程序. 下面是该子程序的代码格式：

```
#if defined(MDL_SET_OUTPUT_PORT_SAMPLE_TIME)
                                  && defined(MATLAB_MEX_FILE)
static void mdlSetOutputPortSampleTime(SimStruct *S,
                                       int_T portIdx,
                                       real_T sampleTime,
                                       real_T offsetTime)
{
}
#endif /* MDL_SET_OUTPUT_PORT_SAMPLE_TIME */
```

3) mdlSetInputPortWidth 子程序. 在向量宽度传播期间，Simulink 为动态端口调用 mdlSetInputPortWidth 子程序，使用假设的宽度. 如果假设的宽度可接受，使用 mdlSetInputPortWidth 子程序设置实际端口的宽度；如果假设的宽度不可接受，则通过 ssSetErrorStatus 产生一个错误. 其代码格式如下：

```
#define MDL_SET_INPUT_PORT_WIDTH /* 改变为 #undef to remove function. */
#if defined(MDL_SET_INPUT_PORT_WIDTH)
                                  && defined(MATLAB_MEX_FILE)
void mdlSetInputPortWidth(SimStruct *S, int portIndex, int width)
{
```

```
}
#endif /* MDL SET_INPUT_PORT_WIDTH */
```

4) mdlSetOutputPortWidth 子程序. 该子程序对于输出端口与 mdlSetInputPortWidth 子程序对于输入端口一样. 其代码形式如下：

```
#define MDL_SET_OUTPUT_PORT_WIDTH /* 变为 #undef to remove function */
#if defined(MDL_SET_OUTPUT_PORT_WIDTH)
                        && defined(MATLAB_MEX_FILE)
void mdlSetOutputPortWidth(SimStruct *S, int portIndex, int width)
{
}
#endif /* MDL SET_OUTPUT_PORT_WIDTH */
```

(8) 为 C MEX 形式 S 函数设置采样时间

Simulink 支持模块执行于不同采样速率. 有两种指定采样速率的方法：一是基于模块的采样时间，二是基于端口的采样时间.

在基于模块的采样时间情况下，如果所有采样时间是最快采样时间的整数倍，S 函数以指定的最快速度来指定模块的所有采样率以及处理输入、输出. 当使用基于端口的采样时间，S 函数对每个输入和输出端口指定采样时间. 为了比较两种方法，分别使用 0.5s 和 0.25s 采样时间. 在基于模块的方法中，选择 0.5 和 0.25，模块将以 0.25s 的增量来执行输入和输出；在基于端口的方法中，可以设置输入端口为 0.5，输出端口为 0.25，模块执行 2Hz 输入 4Hz 输出. 如果应用程序需要输入和输出执行不同的采样率，就使用基于端口的采样时间.

1) 基于模块的采样时间. 使用：ssSetNumSampleTimes(S,numSampleTimes)来配置 S 函数模块为基于模块的采样时间. 其中 numSampleTimes 大于 0，即告知 Simulink，该 S 函数具有基于模块的 S 函数. Simulink 调用 mdlInitializeSampleTimes，设置采样时间.

子程序 mdlInitializeSampleTimes 用来指定两个执行信息：一是采样和偏移时间；在 mdlInitializeSizes 中，使用 ssSetNumSampleTimes 宏来指定 S 函数采样时间，而在子程序 mdlInitializeSampleTimes 中，必须为每个采样时间指定采样周期和偏移量，采样时间可以是输入、输出宽度的函数，也可以指定采样时间为 ssGetInputPortWidth 和 ssGetGetOutputPortWidth 的函数. 二是函数调用：在 ssSetCallSystemOutput 中，指定哪个元素执行函数调用.

采样时间通过使用宏来成对指定为：[采样时间，偏移时间]，这些宏有

```
ssSetSampleTime(S, sampleTimePairIndex, sample_time)
ssSetOffsetTime(S, offsetTimePairIndex, offset_time)
```

其中 sampleTimePairIndex 从 0 开始.

合法的采样时间对有：

```
[CONTINUOUS_SAMPLE_TIME, 0.0 ]
[CONTINUOUS_SAMPLE_TIME, FIXED_IN_MINOR_STEP_OFFSET]
[discrete_sample_period, offset ]
```

[VARIABLE_SAMPLE_TIME , 0.0]

其中的大写值为宏，在 simstruc.h 中定义. 或者可以指定采样时间从驱动模块中继承.

即：[INHERITED_SAMPLE_TIME, 0.0]

或：[INHERITED_SAMPLE_TIME, FIXED_IN_MINOR_STEP_OFFSET]

下面是一个指定两个具有周期为 0.01s 和 0.5s 离散采样时间的例子：

```
static void mdlInitializeSampleTimes(SimStruct *S)
{
  ssSetSampleTime(S, 0, 0.01);
  ssSetOffsetTime(S, 0, 0.0);
  ssSetSampleTime(S, 1, 0.5);
  ssSetOffsetTime(S, 1, 0.0);
} /* End of mdlInitializeSampleTimes. */
```

2）基于端口的采样时间. 使用 ssSetNumSampleTimes(S, PORT_BASED_SAMPLE_TIMES)指定基于端口的采样时间，有

```
ssSetInputPortSampleTime(S, idx, period)
ssSetInputPortOffsetTime(S, idx, offset)
ssSetOutputPortSampleTime(S, idx, period)
ssSetOutputPortOffsetTime(S, idx, offset)
```

inputPortIndex 和 outputPortIndex 范围从 0 至输入(输出)端口数减 1.

当指定基于端口的采样时间，Simulink 就会调用 mdlSetInputPortSampleTime 和 mdlSetOutputPortSampleTime 来确定继承信号的采样率. 一旦所有采样被完全确定，Simulink 就会调用 mdlInitializeSampleTimes 来配置函数调用连接. 如果 S 函数没有任何函数调用连接，该子程序应为空.

当使用基于端口的采样时间时，mdlInitializeSizes 不应包含任何 ssSetSampleTime 或 ssSetOffsetTime 调用.

子程序 mdlSetInputPortSampleTime 的代码样式为

```
#if defined(MDL_SET_INPUT_PORT_SAMPLE_TIME)
                                    && defined(MATLAB_MEX_FILE)
static void mdlSetInputPortSampleTime(SimStruct *S,
                                      int_T portIdx,
                                      real_T sampleTime,
                                      real_T offsetTime)
{
}
#endif /* MDL_SET_INPUT_PORT_SAMPLE_TIME */
```

子程序 mdSetOutputPortSampleTime 的代码样式为

```
#if defined(MDL_SET_OUTPUT_PORT_SAMPLE_TIME)
                                    && defined(MATLAB_MEX_FILE)
static void mdlSetOutputPortSampleTime(SimStruct *S,
```

```
                                    int_T portIdx,
                                    real_T sampleTime,
                                    real_T offsetTime)
{
}
#endif /* MDL_SET_OUTPUT_PORT_SAMPLE_TIME */
```

3) 多采样率 S 函数模块. 在多采样率 S 函数模块中,可以在 mdlOutput 和 mdlUpdate 函数中装入定义每个行为的代码,其中包含一个判断采样点是否发生的语句. 对于一指定的采样时间,ssIsSampleHit 宏确定当前时间是否为采样点. 语法格式为

ssIsSampleHit(S, st_index, tid)

S 为 SimStruct,st_index 指定采样时间索引,tid 为任务代号.

下面是一个对于一个连续模块定义采样时间的例子.

```
/* Initialize the sample time and offset. */
static void mdlInitializeSampleTimes(SimStruct *S)
{
  ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
  ssSetOffsetTime(S, 0, 0.0);
}
```

但必须在 mdlInitializeSizes 函数中加入:ssSetNumSampleTimes(S, 1);

下面是一个定义混合 S 函数模块采样时间的例子:

```
/* Initialize the sample time and offset. */
static void mdlInitializeSampleTimes(SimStruct *S)
{
/* Continuous state sample time and offset. */
  ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
  ssSetOffsetTime(S, 0, 0.0);
/* Discrete state sample time and offset. */
  ssSetSampleTime(S, 1, 0.1);
  ssSetOffsetTime(S, 1, 0.025);
}
```

语句 ssSetNumSampleTimes(S, 2);必须在 mdlInitializeSizes 函数中加入,以指明定义多少个采样时间.

(9) 配置工作向量

如果S函数需要永久的内存存储,可以使用S函数工作向量而不是静态或全局变量. 如果使用静态或全局变量,它们会被S函数的多个实例使用. S函数的跟踪多个实例的这一能力被称为重入.

可以使用工作向量创建一个可重入的S函数. 这些永久性的存储位置是由 Simulink 为S函数管理的. 数据类型可以是整数、浮点数、指针以及一般数据类型. 每一个向量中元素的个数可以被动态地指定为S函数的输入个数的函数.

Simulink 提供了一些宏，它们可用来分配工作向量的各个元素、为各个元素指定变量和访问这些变量.

包含在 mdlInitializeSizes 函数中用来定义工作向量的元素的个数的宏有：

宏 ssSetNumContStates，用来指定连续状态向量宽度；

宏 ssSetNumDiscStates，用来指定离散状态向量宽度；

宏 ssSetNumRWork，用来定义实工作向量宽度；

宏 ssSetNumIWork，用来定义整数工作向量宽度；

宏 ssSetNumPWork，用来定义指针工作向量宽度；

宏 ssSetNumModes，用来定义模式工作向量宽度；

宏 ssSetNumnonsampledZCs，用来定义非采样过零区间向量宽度.

在 mdlInitializeSizes 函数中指定向量宽度，有三种选择：

0，缺省值；表示向量在 S 函数中不使用.

正非 0 整数；这是在仿真循环中要用于 mdlStart，mdlInitializeConditions 和 S 函数子程序的向量宽度.

DYNAMICALLY_SIZED；设置向量宽度为动态变化. Simulink 在传播连线宽度和采样时间后，调整其宽度，一个模块宽度为信号通过的宽度，一般等于输出端口的宽度.

包含在 mdlInitializeConditions 函数中用来设置工作向量的元素的值的宏有：

宏 ssSetRWorkValue(S，rworkIdx，rworkValue)，用来定义实工作向量元素 rworkIdx 的值为 rworkValue.

宏 ssSetIWorkValue(S，iworkIdx，iworkValue)，用来定义整数工作向量元素 iworkIdx 的值为 iworkValue.

宏 ssSetPWorkValue (S，pworkIdx，pworkValue)，用来定义指针工作向量元素 pworkIdx 的值为 pworkValue.

下面这些宏可以取得工作向量的元素的值：

宏 ssGetRWorkValue(S，rworkIdx)，用来获取实工作向量元素 rworkIdx 的值.

宏 ssGetIWorkValue(S，iworkIdx)，用来获取整数工作向量元素 iworkIdx 的值.

宏 ssGetPWorkValue(S，pworkIdx)，用来获取指针工作向量元素 pworkIdx 的值.

可以在 mdlInitializeSizes 函数中指定模块中使用的工作向量的元素的个数. 可以在 mdlInitializeConditions 函数中分配指针工作向量指向的内存，并且在 mdlTerminate 函数中释放它们.

下面的例子打开一个文件并且在指针工作向量中保存 FILE 指针.

包含在 mdlInitializeSizes 函数中如下的这一语句，表明指针工作向量包含有一个元素：

```
ssSetNumPWork(S, 1) /* pointer work vector */
```

下面的代码使用指针工作向量来保存 FILE 指针，FILE 指针是从标准 I/O 函数 fopen 返回的：

```
#define MDL_START /* Change to #undef to remove function. */
#if defined(MDL_START)
static void mdlStart(real_T *x0, SimStruct *S)
```

```
{
  FILE  * fPtr;
  void  * * PWork = ssGetPWork(S);
  fPtr = fopen("file.data", "r");
  PWork[0] = fPtr;
}
#endif /* MDL_START */
```

下面的代码获得指针工作向量中的 FILE 指针并将它传给 fclose 以关闭文件：

```
static void mdlTerminate(SimStruct *S)
{
  if (ssGetPWork(S) != NULL) {
    FILE  * fPtr;
    fPtr = (FILE  * ) ssGetPWorkValue(S,0);
  if (fPtr != NULL) {
     fclose(fPtr);
    }
    ssSetPWorkValue(S,0,NULL);
  }
}
```

如果使用 mdlSetWorkWidths，在 S 函数中使用的任何工作向量，应该在 mdlInitializeSizes 中设置成 DYNAMICALLY_SIZED，即使在 mdlInitializeSizes 调用之前，其值已经知道. S 函数将要用来到的大小，必须在 mdlSetWorkWidths 中指定. 其格式为

```
#define MDL_SET_WORK_WIDTHS /* Change to #undef to remove function.
*/
#if defined(MDL_SET_WORK_WIDTHS) && defined(MATLAB_MEX_FILE)
static void mdlSetWorkWidths(SimStruct  * S)
{
}
#endif /* MDL_SET_WORK_WIDTHS */
```

(10) 内存分配

当创建 S 函数时，可能可用的工作向量不能提供足够的容量. 此时，需要为每个 S 函数实例分配内存. 标准 MATLAB API 内存分配子程序(mxCalloc，mxFree)不能用于 C MEX S 函数. 这些子程序设计从 MATLAB 调用，而不是从 Simulink 调用. 正确的方法是使用 stdlib.h(calloc，free)库子程序来分配内存. 在 mdlStart 中分配并初始化内存，并对指针工作向量元素放置指针：ssGetPWork(S)[i] = ptr；或将赋值为用户数据：ssSetUserData(S，ptr). 在 mdlTerminate 函数中释放分配的内存.

(11) S 函数初始化

S 函数初始化是指初始化 S 函数工作向量或输出信号. 对于 RTW，意味着初始化硬件. 用子程序 mdlStart 和 mdlInitializeConditions 执行初始化.

1）mdlStart 子程序．可选执行子程序 mdlStart 在模型执行开始时调用一次．在其中放置一次初始化代码．其代码格式为

```
#define MDL_START /* 变成 #undef to remove function */
#if defined(MDL_START)
static void mdlStart(SimStruct *S)
{
}
#endif /* MDL_START */
```

2）mdlInitializeConditions 子程序．在可选执行 S 函数子程序 mdlInitializeConditions 中，可以初始化 S 函数模块的连续和离散状态．初始状态放置于状态向量，ssGetContStates(S)和/或 ssGetNumDiscStates(S)．Simulink 将在两处调用该子程序．一是仿真开始；二是在激活子系统中，用来配置复位状态，这种情况下，Simulink 在激活子系统重新执行复位状态时，调用该子程序．可以使用宏 ssIsFirstInitCond(S)来确定当前调用是否是第一次调用．其代码格式为

```
#define MDL_INITIALIZE_CONDITIONS /* 变成 #undef to remove function */
#if defined(MDL_INITIALIZE_CONDITIONS)
static void mdlInitializeConditions(SimStruct *S)
{
}
#endif /* MDL_INITIALIZE_CONDITIONS */
```

下面是一个既有连续又有离散状态的 S 函数例子，它的初始化状态为 1.0．

```
#define MDL_INITIALIZE_CONDITIONS /* 变成 #undef to remove function
*/
#if defined(MDL_INITIALIZE_CONDITIONS)
static void mdlInitializeConditions(SimStruct *S)
{
  int i;
  real_T *xcont = ssGetContStates(S);
  int_T nCStates = ssGetNumContStates(S);
  real_T *xdisc = ssGetRealDiscStates(S);
  int_T nDStates = ssGetNumDiscStates(S);
  for (i = 0; i < nCStates; i++) {
   *xcont++ = 1.0;
  }
  for (i = 0; i < nDStates; i++) {
     *xdisc++ = 1.0;
  }
}
#endif /* MDL_INITIALIZE_CONDITIONS */
```

(12) 仿真期间 S 函数子程序调用

概念上讲，Simulink 执行一个仿真，就是以模型中模块其相连的顺序来执行模块，在执行完这些模块后，再重复这一过程，直到仿真结束. 这种模块的重复执行被称作仿真循环. 在仿真期间 S 函数通过 S 函数子程序与 Simulink 交互.

1) mdlGetTimeOfNextVarHit 子程序. 这是一个可选执行子程序，用于变步长求解器，被调用来获取下一变化采样时间点的时间. 该函数每一主要积分时间步执行一次，它必须使用 ssSetTNext 返回下一采样点. 下一采样点时间必须大于 ssGetT(S).

下一采样点时间可以是输入信号的函数.

下面是该子程序的样式：

```
#define MDL_GET_TIME_OF_NEXT_VAR_HIT /* 变成 #undef to remove function */
#if defined(MDL_GET_TIME_OF_NEXT_VAR_HIT) && \
(defined(MATLAB_MEX_FILE) || defined(NRT))
static void mdlGetTimeOfNextVarHit(SimStruct *S)
{
  ssSetTNext(S, <timeOfNextHit>);
}
#endif /* MDL_GET_TIME_OF_NEXT_VAR_HIT */
```

2) mdlOutputs 子程序. 在 mdlOutputs 子程序中，计算 S 函数模块的输出. 一般输出放入输出向量 ssGetOutputPortSignal 中. 其代码样式为

```
static void mdlOutputs(SimStruct *S, int_T tid)
{
} /* end mdlOutputs */
```

参数 tid 用于与多采样率 S 函数相关联，确定何时一个特定任务有一个采样点.

3) mdlUpdate 子程序. 可选执行子程序 mdlUpdate 每一主要积分时间步调用一次，离散状态一般在此更新，但该函数对于每一积分步发生一次的任何任务都是有用的. 如果在 S 函数模型中包含其代码，就会调用该子程序. 其代码形式为

```
#define MDL_UPDATE /* Change to #undef to remove function. */
#if defined(MDL_UPDATE)
static void mdlUpdate(SimStruct *S, int_T tid)
{
}
#endif /* MDL_UPDATE */
```

当且仅当 S 函数有一或多个离散状态，或者没有直接馈通时，才会调用 S 函数的 mdlUpdate函数. 原因是大多数 S 函数没有离散状态并且有直接馈通而没有更新功能. 因此，在这些情况下 Simulink 能够消除调用另外的程序的需要.

如果 S 函数需要调用它的 mdlUpdate 子程序而不满足前面的任何一种情况，在 mdlInitializeSizes 函数中使用 ssSetNumDiscStates 宏指定它有离散状态.

4) mdlDerivatives 子程序，在可选执行子程序中 mdlDerivatives，计算 S 函数模块连续

状态的导数. 其导数放置于导数向量 ssGetdX(S)中. 子程序 mdlDerivatives 仅在辅助时间步期间调用. 每次调用该子程序,必须明确设置所有导数的值. 导数向量不从上次调用子程序中维持其值. 分配给导数向量的内存在执行期间是改变的. 其代码形式为

```
#define MDL_DERIVATIVES /* Change to #undef to remove function. */
#if defined(MDL_DERIVATIVES)
static void mdlDerivatives(SimStruct *S)
{
}
#endif /* MDL_DERIVATIVES */
```

5) mdlZeroCrossings 子程序. 当 S 函数注册 CONTINUOUS_SAMPLE_TIME,并且具有非采样过零区间(ssGetNumNonsampledZCs(S) > 0),可以使用可选执行子程序 mdlZeroCrossings. 该子程序用于提供 Simulink 信号的过零区间. 提供给 Simulink 过零信号,子程序 mdlZeroCrossings 就会更新 ssGetNumNonsampledZCs(S)向量. 其代码格式为

```
#define MDL_ZERO_CROSSINGS /* Change to #undef to remove function. */
#if defined(MDL_ZERO_CROSSINGS)
                        && (defined(MATLAB_MEX_FILE) || defined(NRT))
static void mdlZeroCrossings(SimStruct *S)
{
}
#endif /* MDL_ZERO_CROSSINGS */
```

6) mdlTerminate 子程序. 在 mdlTerminate 子程序中,执行结束仿真所需的所有动作. 如在 mdlInitializeSizes 或 mdlStart 中分配的内存,此时要释放. 假设 S 函数分配了一些碎块内存,并将其存于 Pwork 中,下面的代码可以释放其内存.

```
static void mdlTerminate(SimStruct *S)
{
  int i;
  for (i = 0; i<ssGetNumPWork(S); i++) {
    if (ssGetPWorkValue(S,i) != NULL) {
     free(ssGetPWorkValue(S,i));
    }
  }
}
```

(13) 在实时工作空间中使用 S 函数

一般来讲,在 RTW 中可以使用 S 函数,但在有些情况下,需要对 S 函数进行修改.

1) RTW 编译的 S 函数模块名. 如果 S 函数与多个模块连编在一起,必须为其它模块提供编译过程名字. 可以使用 RTW 模板构成文件的方法,或更方便使用 MATLAB 的 set_param 命令. 如与其它模块连编在一起:

```
mex sfun_main.c sfun_module1.c sfun_module2.c
```

指定模块名字使用 MATLAB 命令,无需扩展名如下:

```
set_param(sfun_block,'SFunctionModules','sfun_module1 sfun_module2')
```

参数也可以是变量，如：

```
modules = 'sfun_module1 sfun_module2'
set_param(sfun_block,'SFunctionModules','modules')
```

参数还可以是一个被求的字符串，如：

```
set_param(sfun_block,'SFunctionModules','''sfun_module1 sfun_module2''')
```

2）使用 RTW 产生代码的 S 函数 RTWdata. 模块有一个属性，被称为 RTWdata，当嵌入 S 函数时，可用于目标语言编译器. RTWdata 是一个字符串结构，可用于连接模块，在产生代码时，它与模型一起保存，并放在 model. rtw 文件中. 如下面一系列命令：

```
mydata.field1 = 'information for field1';
mydata.field2 = 'information for field2';
set_param(gcb,'RTWdata',mydata)
get_param(gcb,'RTWdata')
```

产生如下结果：

```
ans =
field1: 'information for field1'
field2: 'information for field2'
```

对于相关的 S 函数模块在 model. rtw 中有如下的信息：

```
Block {
  Type   "S-Function"
  RTWdata {
    field1 "information for field1"
    field2 "information for field2"
  }
}
```

3）mdlRTW 子程序. 在产生代码时，该子程序可以帮助嵌入 S 函数. 由于性能的原因，如果有一个 mdlProcessParameters 子程序，或者 S 函数具有一个“仿真模式”和一个“实时模式”，也需要嵌入 S 函数，如：一个硬件输入/输出 S 函数，即 Simulink I/O 设备与实时 I/O 设备交互的仿真.

在生成的代码中嵌入 S 函数，必须使用目标语言编译器.

9.3.3 C MEX 文件形式的 S 函数例子

9.3.3.1 简单例子

图 9.12 所示的例子将输入信号的幅值翻倍，它是用 C MEX 文件形式的格式写成的.

包含该 S 函数的 C 代码是基于一个被称为 sfuntmpl. c 的 Simulink 的 S 函数模板写成的. 使用这一模板，可以用它所提供的结构创建一个 C 函数. 下面是 timestwo. c 中的 MATLAB 代码：

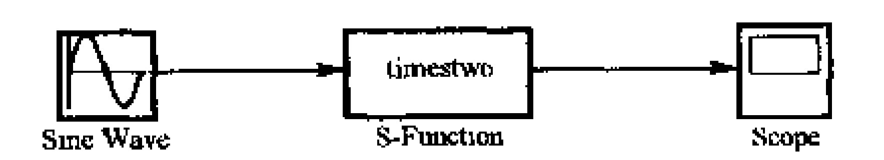


图 9.12 使用 S 函数的示例模型

```
/*
* File : timestwo.c
* Abstract:
*         An example C-file S-function for multiplying an input by 2,
*         y  = 2*u
*
* See simulink/src/sfuntmpl.doc
*
* Copyright (c) 1990-1998 by The MathWorks, Inc. All Rights Reserved.
*  $Revision: 1.3 $
*/
#define S_FUNCTION_NAME   timestwo
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
/* Function: mdlInitializeSizes ===================
 * Abstract:
 *   Setup sizes of the various vectors.
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, 0);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return; /* Parameter mismatch will be reported by Simulink */
    }
    if (! ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, DYNAMICALLY_SIZED);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    if (! ssSetNumOutputPorts(S,1)) return;
    ssSetOutputPortWidth(S, 0, DYNAMICALLY_SIZED);
    ssSetNumSampleTimes(S, 1);
    /* Take care when specifying exception free code - see sfuntmpl.doc */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}
/* Function: mdlInitializeSampleTimes =====================
 * Abstract:
```

```
 *      Specifiy that we inherit our sample time from the driving block.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}
/* Function: mdlOutputs =====================
 * Abstract:
 *      y = 2*u
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
    int_T               i;
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    real_T              *y      = ssGetOutputPortRealSignal(S,0);
    int_T               width = ssGetOutputPortWidth(S,0);
    for (i=0; i<width; i++) {
        *y++ = 2.0 *(*uPtrs[i]);
    }
}
/* Function: mdlTerminate =====================
 * Abstract:
 *      No termination needed, but we are required to have this routine.
 */
static void mdlTerminate(SimStruct *S)
{
}
#ifdef  MATLAB_MEX_FILE   /* Is this file being compiled as a MEX-file? */
#include "simulink.c"       /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"        /* Code generation registration function */
#endif
```

M 文件和 C MEX 文件形式的 S 函数之间一个最大的区别是 C MEX 文件形式的 S 函数必须有对 mdlUpdate 和 mdlDerivative 的调用,即使模型中没有连续或者离散状态.没有这些函数,MEX 文件是不能成功地被编译通过的.

通常,大多数 S 函数模块需要对状态(连续或离散)进行处理.在 Simulink 中用 S 函数可以建模的系统类型包括:连续、离散、混合、变步长采样时间、过零区间、时间变化连续传递函数.

下面所有的例子都是基于 C MEX 文件形式的 S 函数的模板文件 sfuntmpl.c 和 sfuntmpl.doc 生成的.

9.3.3.2　连续状态 S 函数

下例说明如何用 C 语言对一列连续状态方程进行建模.该例的文件名是 csfunc.c,位于目录 simulink/src 下:

```
/*   File     : csfunc.c
 *   Abstract:
 *
 *        Example C-file S-function for defining a continuous system.
 *
 *        x' = Ax + Bu
 *        y  = Cx + Du
 *
 *        For more details about S-functions, see simulink/src/sfuntmpl.doc.
 *
 *   Copyright (c) 1990-1998 by The MathWorks, Inc. All Rights Reserved.
 *   $Revision: 1.3 $
 */
#define S_FUNCTION_NAME csfunc
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#define U(element) (*uPtrs[element])  /* Pointer to Input Port0 */
static real_T A[2][2]={ { -0.09, -0.01 } ,
                        {  1   ,  0    }
                      };
static real_T B[2][2]={ {  1   , -7    } ,
                        {  0   , -2    }
                      };
static real_T C[2][2]={ {  0   , 2     } ,
                        {  1   , -5    }
                      };
static real_T D[2][2]={ { -3   , 0     } ,
                        {  1   , 0     }
                      };
/*======================*
 * S-function methods *
 *======================*/
/* Function: mdlInitializeSizes ================
```

```
 * Abstract:
 *     The sizes information is used by Simulink to determine the S-function
 *     block's characteristics (number of inputs, outputs, states, etc.).
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, 0);  /* Number of expected parameters */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return; /* Parameter mismatch will be reported by Simulink */
    }
    ssSetNumContStates(S, 2);
    ssSetNumDiscStates(S, 0);
    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 2);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 2);
    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
    /* Take care when specifying exception free code - see sfuntmpl.doc */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}
/* Function: mdlInitializeSampleTimes ====================
 * Abstract:
 *     Specifiy that we have a continuous sample time.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}
#define MDL_INITIALIZE_CONDITIONS
/* Function: mdlInitializeConditions ====================
 * Abstract:
 *     Initialize both continuous states to zero.
```

```
 */
static void mdlInitializeConditions(SimStruct  *S)
{
    real_T  *x0 = ssGetContStates(S);
    int_T   lp;
    for (lp=0;lp<2;lp++) {
        *x0++=0.0;
    }
}
/* Function: mdlOutputs =========================
 * Abstract:
 *       y = Cx + Du
 */
static void mdlOutputs(SimStruct  *S, int_T tid)
{
    real_T              *y    = ssGetOutputPortRealSignal(S,0);
    real_T              *x    = ssGetContStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);

    /* y=Cx+Du */
    y[0]=C[0][0]*x[0]+C[0][1]*x[1]+D[0][0]*U(0)+D[0][1]*U(1);
    y[1]=C[1][0]*x[0]+C[1][1]*x[1]+D[1][0]*U(0)+D[1][1]*U(1);
}
#define MDL_DERIVATIVES
/* Function: mdlDerivatives =================
 * Abstract:
 *       xdot = Ax + Bu
 */
static void mdlDerivatives(SimStruct  *S)
{
    real_T              *dx   = ssGetdX(S);
    real_T              *x    = ssGetContStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    /* xdot=Ax+Bu */
    dx[0]=A[0][0]*x[0]+A[0][1]*x[1]+B[0][0]*U(0)+B[0][1]*
U(1);
    dx[1]=A[1][0]*x[0]+A[1][1]*x[1]+B[1][0]*U(0)+B[1][1]*
U(1);
}
```

```
/* Function: mdlTerminate =====================
 * Abstract:
 *    No termination needed, but we are required to have this routine.
 */
static void mdlTerminate(SimStruct *S)
{
}
#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file?
*/
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

该 MEX 文件是 csfunc.m 文件的 C 语言版本.状态向量 x 的导数的计算是在 mdlDerivative 函数中,输出 y 的计算是在 mdlOutputs 函数中.因为没有离散状态并且在结束时没有任务需要完成,所以没有 mdlUpdate,mdlTerminate 是空的.

9.3.3.3 离散状态 S 函数

Simulink 包含一个名为 dsfunc.c 的函数,它是一个用 S 函数建模的离散状态系统.下面是该 S 函数的 C 语言代码:

```
/*  File    : dsfunc.c
 *  Abstract:
 *
 *     Example C-file S-function for defining a discrete system.
 *
 *     x(n+1) = Ax(n) + Bu(n)
 *     y(n)   = Cx(n) + Du(n)
 *
 *     For more details about S-functions, see simulink/src/sfuntmpl.doc.
 *
 *  Copyright (c) 1990-1998 by The MathWorks, Inc. All Rights Reserved.
 *   $Revision: 1.4 $
 */
#define S_FUNCTION_NAME dsfunc
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#define U(element) (*uPtrs[element])  /* Pointer to Input Port0 */
static real_T A[2][2]={ { -1.3839, -0.5097 } ,
                        { 1      , 0       }
```

```
                       };
static real_T B[2][2]={ { -2.5559,  0       },
                        { 0      ,  4.2382 }
                       };
static real_T C[2][2]={ { 0      ,  2.0761 },
                        { 0      ,  7.7891 }
                       };
static real_T D[2][2]={ { -0.8141, -2.9334 },
                        { 1.2426,  0       }
                       };
/*======================*
 * S-function methods *
 *======================*/
/* Function: mdlInitializeSizes ================
 * Abstract:
 *    The sizes information is used by Simulink to determine the S-function
 *    block's characteristics (number of inputs, outputs, states, etc.).
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, 0);  /* Number of expected parameters */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return; /* Parameter mismatch will be reported by Simulink */
    }
    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 2);
    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 2);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 2);
    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
    /* Take care when specifying exception free code - see sfuntmpl.doc */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
```

```
}
/* Function: mdlInitializeSampleTimes ===============
 * Abstract:
 *     Specifiy that we inherit our sample time from the driving block.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, 1.0);
    ssSetOffsetTime(S, 0, 0.0);
}
#define MDL_INITIALIZE_CONDITIONS
/* Function: mdlInitializeConditions ===================
 * Abstract:
 *     Initialize both continuous states to zero.
 */
static void mdlInitializeConditions(SimStruct *S)
{
    real_T *x0 = ssGetRealDiscStates(S);
    int_T  lp;
    for (lp=0;lp<2;lp++) {
        *x0++=1.0;
    }
}
/* Function: mdlOutputs ========================
 * Abstract:
 *        y = Cx + Du
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
    real_T              *y    = ssGetOutputPortRealSignal(S,0);
    real_T              *x    = ssGetRealDiscStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    /* y=Cx+Du */
    y[0]=C[0][0]*x[0]+C[0][1]*x[1]+D[0][0]*U(0)+D[0][1]*U(1);
    y[1]=C[1][0]*x[0]+C[1][1]*x[1]+D[1][0]*U(0)+D[1][1]*U(1);
}
#define MDL_UPDATE
/* Function: mdlUpdate ========================
 * Abstract:
```

```
 *        xdot = Ax + Bu
 */
static void mdlUpdate(SimStruct *S, int_T tid)
{
    real_T              tempX[2] = {0.0, 0.0};
    real_T              *x        = ssGetRealDiscStates(S);
    InputRealPtrsType uPtrs     = ssGetInputPortRealSignalPtrs(S,0);
    /* xdot=Ax+Bu */
     tempX[0]=A[0][0]*x[0]+A[0][1]*x[1]+B[0][0]*U(0)+B[0][1]*
U(1);
     tempX[1]=A[1][0]*x[0]+A[1][1]*x[1]+B[1][0]*U(0)+B[1][1]*
U(1);
    x[0]=tempX[0];
    x[1]=tempX[1];
}
/* Function: mdlTerminate ========================
 * Abstract:
 *      No termination needed, but we are required to have this routine.
 */
static void mdlTerminate(SimStruct *S)
{
}
#ifdef   MATLAB_MEX_FILE      /* Is this file being compiled as a MEX-file?
*/
#include "simulink.c"        /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"         /* Code generation registration function */
#endif
```

上面是 dsfunc.m 的 C 语言的等效形式.对离散状态向量 x 的更新是在 mdlUpdate 函数中,对输出的计算是在 mdlOutputs 函数中.因为没有连续状态并且在结束时没有任务需要完成,所以没有 mdlDerivatives,mdlTerminate 都是空函数.

9.3.3.4 混合系统 S 函数的例子

Simulink 包含一个名为 mixedm.c 的函数,它是一个用 S 函数 mixedm.c 建模的混合系统(连续状态和离散状态的组合)的例子.mixedm.c 组合了 csfunc.c 和 dsfunc.c 中的元素.如果需要对一个混合模型建模,所要做的就是在 mdlDerivative 中放进该模型的连续方程,在 mdlUpdate 中放进模型的离散方程.

下面以图 9.6 所示的模型为例.

该模型用 C 语言写成的 S 函数保存于 mixedm.c 中,它实现了一个连续的积分器后

跟一个离散的单位延迟，代码如下：

```
/*  File    : mixedm.c
 *  Abstract:
 *
 *  An example S-function illustrating multiple sample times by implementing
 *          integrator -> ZOH(Ts=1second) -> UnitDelay(Ts=1second)
 *      with an initial condition of 1.
 *  (e.g. an integrator followed by unit delay operation).
 *
 *      For more details about S-functions, see simulink/src/sfuntmpl.doc
 *
 *  Copyright (c) 1990-1998 by The MathWorks, Inc. All Rights Reserved.
 *   $Revision: 1.7 $
 */
#define S_FUNCTION_NAME mixedm
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#define U(element) (*uPtrs[element])  /* Pointer to Input Port0 */
/*======================*
 * S-function methods *
 *======================*/
/* Function: mdlInitializeSizes =======================
 * Abstract:
 *    The sizes information is used by Simulink to determine the S-function
 *    block's characteristics (number of inputs, outputs, states, etc.).
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, 0);  /* Number of expected parameters */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return; /* Parameter mismatch will be reported by Simulink */
    }
    ssSetNumContStates(S, 1);
    ssSetNumDiscStates(S, 1);
    ssSetNumRWork(S, 1);  /* for zoh output feeding the delay operator */
    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 1);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
```

```
    ssSetInputPortOffsetTime(S, 0, 0.0);
    if (! ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 1);
    ssSetOutputPortSampleTime(S, 0, 1.0);
    ssSetOutputPortOffsetTime(S, 0, 0.0);
    ssSetNumSampleTimes(S, 2);
    /* Take care when specifying exception free code - see sfuntmpl.doc. */
    ssSetOptions(S, (SS_OPTION_EXCEPTION_FREE_CODE |
                SS_OPTION_PORT_SAMPLE_TIMES_ASSIGNED));
} /* end mdlInitializeSizes */
/* Function: mdlInitializeSampleTimes ===============
 * Abstract:
 *     Two tasks: One continuous, one with discrete sample time of 1.0.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
    ssSetSampleTime(S, 1, 1.0);
    ssSetOffsetTime(S, 1, 0.0);
} /* end mdlInitializeSampleTimes */
#define MDL_INITIALIZE_CONDITIONS
/* Function: mdlInitializeConditions ================
 * Abstract:
 *     Initialize both continuous states to one.
 */
static void mdlInitializeConditions(SimStruct *S)
{
    real_T *xC0 = ssGetContStates(S);
    real_T *xD0 = ssGetRealDiscStates(S);
    xC0[0] = 1.0;
    xD0[0] = 1.0;
} /* end mdlInitializeConditions */
/* Function: mdlOutputs ====================
 * Abstract:
 *       y = xD, and update the zoh internal output.
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
```

```
    /* update the internal "zoh" output */
    if (ssIsContinuousTask(S, tid)) {
        if (ssIsSpecialSampleHit(S, 1, 0, tid)) {
            real_T *zoh = ssGetRWork(S);
            real_T *xC  = ssGetContStates(S);
            *zoh = *xC;
        }
    }
    /* y=xD */
    if (ssIsSampleHit(S, 1, tid)) {
        real_T *y   = ssGetOutputPortRealSignal(S,0);
        real_T *xD  = ssGetRealDiscStates(S);
        y[0]=xD[0];
    }
} /* end mdlOutputs */
#define MDL_UPDATE
/* Function: mdlUpdate =====================
 * Abstract:
 *       xD = xC
 */
static void mdlUpdate(SimStruct *S, int_T tid)
{
    /* xD=xC */
    if (ssIsSampleHit(S, 1, tid)) {
        real_T *xD = ssGetRealDiscStates(S);
        real_T *zoh = ssGetRWork(S);
        xD[0]= *zoh;
    }
} /* end mdlUpdate */
#define MDL_DERIVATIVES
/* Function: mdlDerivatives =================
 * Abstract:
 *       xdot = U
 */
static void mdlDerivatives(SimStruct *S)
{
    real_T              *dx    = ssGetdX(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    /* xdot=U */
```

```
    dx[0]=U(0);
} /* end mdlDerivatives */
/* Function: mdlTerminate ================================
 * Abstract:
 *    No termination needed, but we are required to have this routine.
 */
static void mdlTerminate(SimStruct *S)
{
}
#ifdef  MATLAB_MEX_FILE     /* Is this file being compiled as a MEX-file?
*/
#include "simulink.c"       /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"        /* Code generation registration function */
#endif
```

因为在结束时没有任务需要完成,所以 mdlTerminate 是一个空函数. mdlDerivatives 计算连续状态向量 x 的导数,而 mdlUpdate 包含有用来更新离散状态 x 的方程.

9.3.3.5 变步长 S 函数的例子

该例是一个使用变步长的 S 函数. 变步长的函数需要调用 mdlGetTimeOfNextVarHit,它是一个用来计算下一个采样时间点的 S 函数程序.

该 S 函数是一个名为 vsfunc.c 的 C 语言程序. 它是一个离散的 S 函数,它的第一个输入延迟时间由第二个输入决定. 下面是 vsfunc.c 的 C 代码:

```
/*  File   : vsfunc.c
 *  Abstract:
 *
 *      Example C-file S-function for defining a continuous system.
 *
 *      Variable step S-function example.
 *      This example S-function illustrates how to create a variable step
 *      block in Simulink.  This block implements a variable step delay
 *      in which the first input is delayed by an amount of time determined
 *      by the second input:
 *
 *      dt      = u(2)
 *      y(t+dt) = u(t)
 *
 *      For more details about S-functions, see simulink/src/sfuntmpl.doc.
 *
```

```
*   Copyright (c) 1990—1998 by The MathWorks, Inc. All Rights Reserved.
*   $Revision: 1.6 $
*/
#define S_FUNCTION_NAME vsfunc
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#define U(element) (*uPtrs[element])  /* Pointer to Input Port0 */
/* Function: mdlInitializeSizes ================================
 * Abstract:
 *     The sizes information is used by Simulink to determine the S-function
 *     block's characteristics (number of inputs, outputs, states, etc.).
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, 0);  /* Number of expected parameters */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return; /* Parameter mismatch will be reported by Simulink */
    }
    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 1);
    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 2);
    ssSetInputPortDirectFeedThrough(S, 0, 0);
    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 1);
    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
    /* Take care when specifying exception free code - see sfuntmpl.doc */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}
/* Function: mdlInitializeSampleTimes ====================================
 * Abstract:
 *     Variable-Step S-function
 */
static void mdlInitializeSampleTimes(SimStruct *S)
```

```
{
    ssSetSampleTime(S, 0, VARIABLE_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}
#define MDL_INITIALIZE_CONDITIONS
/* Function: mdlInitializeConditions =================
 * Abstract:
 *     Initialize discrete state to zero.
 */
static void mdlInitializeConditions(SimStruct *S)
{
    real_T *x0 = ssGetRealDiscStates(S);
    x0[0] = 0.0;
}
#define MDL_GET_TIME_OF_NEXT_VAR_HIT
static void mdlGetTimeOfNextVarHit(SimStruct *S)
{
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    /* Make sure input will increase time */
    if (U(1) <= 0.0) {
        /* If not, abort simulation */
        ssSetErrorStatus(S,"Variable step control input must be "
                         "greater than zero");
        return;
    }
  ssSetTNext(S, ssGetT(S)+U(1));
}
/* Function: mdlOutputs ======================
 * Abstract:
 *      y = x
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
    real_T *y = ssGetOutputPortRealSignal(S,0);
    real_T *x = ssGetRealDiscStates(S);
    /* Return the current state as the output */
    y[0] = x[0];
}
#define MDL_UPDATE
```

```
/* Function: mdlUpdate ========================
 * Abstract:
 *     This function is called once for every major integration time step.
 *     Discrete states are typically updated here, but this function is useful
 *     for performing any tasks that should only take place once per integration
 *     step.
 */
static void mdlUpdate(SimStruct *S, int_T tid)
{
    real_T              *x    = ssGetRealDiscStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    x[0]=U(0);
}
/* Function: mdlTerminate =====================
 * Abstract:
 *     No termination needed, but we are required to have this routine.
 */
static void mdlTerminate(SimStruct *S)
{
}
#ifdef  MATLAB_MEX_FILE     /* Is this file being compiled as a MEX-file?
*/
#include "simulink.c"       /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"        /* Code generation registration function */
#endif
```

其中 vsfunc 的输出仅仅是输入 u 延迟一段可变的时间之后的结果. 在 mdlOutputs 中将输出 y 设为等于状态 x. 在 mdlUpdate 中设置状态 x 等于输入向量 u. 该例子调用 mdlGetTimeOfNextVarHit,它是一个用来计算并设定“下一个点的时间”(vsfunc 下一次被调用的时间)的 S 函数子程序. 在 mdlGetTimeOfNextVarHit 中宏 ssGetU 用来获得指向输入 u 的指针. 然后该调用在下面的语句中实现:

```
ssSetTNext(S, ssGetT(S)+u[1]);
```

宏 ssGetT 取得仿真时间 t. 模块的第二个输入 u[1]被用来加给 t,并且宏 ssSetTNext 设置下一个点的时间等于 t+u[1],它使输出延迟一段由 u[1]设定的时间.

9.3.3.6 过零 S 函数

S 函数例子中 sfun_zc_sat 演示了如何实现一个饱和模块,该 S 函数设计既具有定步长又具有变步求解器. 当其继承采样时间,就使用变步长求解器,过零区间算法用来准确确定饱和何时发生. 下面是其 C 代码.

```
/*  File    : sfun_zc_sat.c
 *  Abstract:
 *
 *      Example of an S-function which has nonsampled zero crossings to
 *      implement a saturation function.  This S-function is designed to be
 *      used with a variable or fixed step solver.
 *
 *  A saturation is described by three equations
 *
 *    (1)     y = UpperLimit
 *    (2)     y = u
 *    (3)     y = LowerLimit
 *
 *  and a set of inequalities that specify which equation to use
 *
 *    if                          UpperLimit < u    then   use (1)
 *    if       LowerLimit <= u <= UpperLimit          then   use (2)
 *    if   u < LowerLimit                           then   use (3)
 *
 *  A key fact is that the valid equation 1, 2, or 3, can change at
 *  any instant.  Nonsampled zero crossing support helps the variable step
 *  solvers locate the exact instants when behavior switches from one equation
 *  to another.
 *
 *  Copyright (c) 1990-1998 by The MathWorks, Inc. All Rights Reserved.
 *   $Revision: 1.6 $
 */
#define S_FUNCTION_NAME  sfun_zc_sat
#define S_FUNCTION_LEVEL 2
#include "tmwtypes.h"
#include "simstruc.h"
#ifdef MATLAB_MEX_FILE
# include "mex.h"
#endif
/*=========================================*
 * General Defines/macros *
 *=========================================*/
/* index to Upper Limit */
#define I_PAR_UPPER_LIMIT 0
```

```
/* index to Upper Limit */
#define I_PAR_LOWER_LIMIT 1
/* total number of block parameters */
#define N_PAR                    2
/*
 *  Make access to mxArray pointers for parameters more readable.
 */
#define P_PAR_UPPER_LIMIT ( ssGetSFcnParam(S,I_PAR_UPPER_LIMIT) )
#define P_PAR_LOWER_LIMIT(ssGetSFcnParam(S,I_PAR_LOWER_LIMIT) )
#define      MDL_CHECK_PARAMETERS
#if defined(MDL_CHECK_PARAMETERS) && defined(MATLAB_MEX_FILE)
  /* Function: mdlCheckParameters ==================
   * Abstract:
   *   Check that parameter choices are allowable.
   */
  static void mdlCheckParameters(SimStruct *S)
  {
      int_T       i;
      int_T       numUpperLimit;
      int_T       numLowerLimit;
      const char *msg = NULL;
      /*
       * check parameter basics
       */
      for ( i = 0; i < N_PAR; i++ ) {
          if ( mxIsEmpty(     ssGetSFcnParam(S,i) ) ||
               mxIsSparse(    ssGetSFcnParam(S,i) ) ||
               mxIsComplex(   ssGetSFcnParam(S,i) ) ||
               ! mxIsNumeric( ssGetSFcnParam(S,i) ) ) {
              msg = "Parameters must be real vectors.";
              goto EXIT_POINT;
          }
      }
      /*
       * Check sizes of parameters.
       */
      numUpperLimit = mxGetNumberOfElements( P_PAR_UPPER_LIMIT );
      numLowerLimit = mxGetNumberOfElements( P_PAR_LOWER_LIMIT );
      if ( ( numUpperLimit != 1              ) &&
```

```
            ( numLowerLimit != 1              ) &&
            ( numUpperLimit != numLowerLimit ) ) {
           msg = "Number of input and output values must be equal.";
           goto EXIT_POINT;
       }
       /*
        * Error exit point
        */
  EXIT_POINT:
       if (msg != NULL) {
           ssSetErrorStatus(S, msg);
       }
  }
#endif /* MDL_CHECK_PARAMETERS */
/* Function: mdlInitializeSizes =====================
 * Abstract:
 *    Initialize the sizes array.
 */
static void mdlInitializeSizes(SimStruct *S)
{
    int_T numUpperLimit, numLowerLimit, maxNumLimit;
    /*
     * Set and Check parameter count
     */
    ssSetNumSFcnParams(S, N_PAR);
#if defined(MATLAB_MEX_FILE)
    if (ssGetNumSFcnParams(S) == ssGetSFcnParamsCount(S)) {
        mdlCheckParameters(S);
        if (ssGetErrorStatus(S) != NULL) {
            return;
        }
    } else {
        return; /* Parameter mismatch will be reported by Simulink */
    }
#endif
    /*
     * Get parameter size info.
     */
    numUpperLimit = mxGetNumberOfElements( P_PAR_UPPER_LIMIT );
```

```
numLowerLimit = mxGetNumberOfElements( P_PAR_LOWER_LIMIT );
if (numUpperLimit > numLowerLimit) {
    maxNumLimit = numUpperLimit;
} else {
    maxNumLimit = numLowerLimit;
}
/*
 * states
 */
ssSetNumContStates(S, 0);
ssSetNumDiscStates(S, 0);
/*
 * outputs
 *   The upper and lower limits are scalar expanded
 *   so their size determines the size of the output
 *   only if at least one of them is not scalar.
 */
if (! ssSetNumOutputPorts(S, 1)) return;
if ( maxNumLimit > 1 ) {
    ssSetOutputPortWidth(S, 0, maxNumLimit);
} else {
    ssSetOutputPortWidth(S, 0, DYNAMICALLY_SIZED);
}
/*
 * inputs
 *   If the upper or lower limits are not scalar then
 *   the input is set to the same size.  However, the
 *   ssSetOptions below allows the actual width to
 *   be reduced to 1 if needed for scalar expansion.
 */
if (! ssSetNumInputPorts(S, 1)) return;
ssSetInputPortDirectFeedThrough(S, 0, 1 );
if ( maxNumLimit > 1 ) {
    ssSetInputPortWidth(S, 0, maxNumLimit);
} else {
    ssSetInputPortWidth(S, 0, DYNAMICALLY_SIZED);
}
/*
 * sample times
```

```
    */
   ssSetNumSampleTimes(S, 1);
   /*
    * work
    */
   ssSetNumRWork(S, 0);
   ssSetNumIWork(S, 0);
   ssSetNumPWork(S, 0);
   /*
    * Modes and zero crossings:
    * If we have a variable step solver and this block has a continuous
    * sample time, then
    *    o One mode element will be needed for each scalar output
    *      in order to specify which equation is valid (1), (2), or (3).
    *    o Two ZC elements will be needed for each scalar output
    *      in order to help the solver find the exact instants
    *      at which either of the two possible "equation switches"
    *      One will be for the switch from eq. (1) to (2);
    *      the other will be for eq. (2) to (3) and vise versa.
    * otherwise
    *    o No modes and nonsampled zero crossings will be used.
    *
    */
   ssSetNumModes(S, DYNAMICALLY_SIZED);
   ssSetNumNonsampledZCs(S, DYNAMICALLY_SIZED);
   /*
    * options
    *    o No mexFunctions and no problematic mxFunctions are called
    *      so the exception free code option safely gives faster simulations.
    *    o Scalar expansion of the inputs is desired.  The option provides
    *      this without the need to  write mdlSetOutputPortWidth and
    *      mdlSetInputPortWidth functions.
    */
   ssSetOptions(S, ( SS_OPTION_EXCEPTION_FREE_CODE |

SS_OPTION_ALLOW_INPUT_SCALAR_EXPANSION));
} /* end mdlInitializeSizes */
/* Function: mdlInitializeSampleTimes ================
 * Abstract:
```

```
 *     Specify that the block is continuous.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0);
}
#define    MDL_SET_WORK_WIDTHS
#if defined(MDL_SET_WORK_WIDTHS) && defined(MATLAB_MEX_FILE)
/* Function: mdlSetWorkWidths =====================
 *   The width of the Modes and the ZCs depends on the width of the output.
 *   This width is not always known in mdlInitializeSizes so it is handled
 *   here.
 */
static void mdlSetWorkWidths(SimStruct *S)
{
    int nModes;
    int nNonsampledZCs;
    if (ssIsVariableStepSolver(S) &&
        ssGetSampleTime(S,0) == CONTINUOUS_SAMPLE_TIME &&
        ssGetOffsetTime(S,0) == 0.0) {
        int numOutput = ssGetOutputPortWidth(S, 0);
        /*
         * modes and zero crossings
         *     o One mode element will be needed for each scalar output
         *       in order to specify which equation is valid (1), (2), or (3).
         *     o Two ZC elements will be needed for each scalar output
         *       in order to help the solver find the exact instants
         *       at which either of the two possible "equation switches"
         *       One will be for the switch from eq. (1) to (2);
         *       the other will be for eq. (2) to (3) and vise-versa.
         */
        nModes         = numOutput;
        nNonsampledZCs = 2 * numOutput;
    } else {
        nModes         = 0;
        nNonsampledZCs = 0;
    }
    ssSetNumModes(S,nModes);
```

```
    ssSetNumNonsampledZCs(S,nNonsampledZCs);
}
#endif /* MDL_SET_WORK_WIDTHS */
/* Function: mdlOutputs ===========================================
 * Abstract:
 *
 *   A saturation is described by three equations
 *
 *      (1)      y = UpperLimit
 *      (2)      y = u
 *      (3)      y = LowerLimit
 *
 *   When this block is used with a fixed-step solver or it has a noncontinuous
 *   sample time, the equations are used as it
 *
 *   Now consider the case of this block being used with a variable step solver
 *   and it has a continusous sample time. Solvers work best on smooth problems.
 *   In order for the solver to work without chattering, limit cycles, or
 *   similar problems.  It is absolutely crucial that the same equation be used
 *   throughout the duration of a MajorTimeStep. To visualize this, consider
 *   the case of the Saturation block feeding an Integrator block.
 *
 *   To implement this rule, the mode vector is used to specify the
 *   valid equation based on the following:
 *
 *     if                              UpperLimit < u      then    use (1)
 *     if         LowerLimit <= u <= UpperLimit            then    use (2)
 *     if    u < LowerLimit                                then    use (3)
 *
 *   The mode vector is changed only at the beginning of a MajorTimeStep.
 *
 *   During a minor time step, the equation specified by the mode vector
 *   is used without question.   Most of the time, the value of u will agree
 *   with the equation specified by the mode vector.   However, sometimes u's
 *   value will indicate a different equation.   Nonetheless, the equation
 *   specified by the mode vector must be used.
 *
 *   When the mode and u indicate different equations, the corresponding
 *   calculations are not correct.   However, this is not a problem.   From
```

```
 *  the ZC function, the solver will know that an equation switch occured
 *  in the middle of the last MajorTimeStep.  The calculations for that
 *  time step will be discarded.  The ZC function will help the solver
 *  find the exact instant at which the switch occured.  Using this knowledge,
 *  the length of the MajorTimeStep will be reduced so that only one equation
 *  is valid throughout the entire time step.
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
    InputRealPtrsType uPtrs      = ssGetInputPortRealSignalPtrs(S,0);
    real_T            *y         = ssGetOutputPortRealSignal(S,0);
    int_T             numOutput  = ssGetOutputPortWidth(S,0);
    int_T             iOutput;
    /*
     * Set index and increment for input signal, upper limit, and lower limit
     * parameters so that each gives scalar expansion if needed.
     */
    int_T  uIdx          = 0;
    int_T  uInc          = ( ssGetInputPortWidth(S,0) > 1 );
    real_T *upperLimit   = mxGetPr( P_PAR_UPPER_LIMIT );
    int_T  upperLimitInc = ( mxGetNumberOfElements( P_PAR_UPPER_LIM-
IT ) > 1 );
    real_T *lowerLimit   = mxGetPr( P_PAR_LOWER_LIMIT );
    int_T  lowerLimitInc = ( mxGetNumberOfElements( P_PAR_LOWER_LIM-
IT ) > 1 );
    if (ssGetNumNonsampledZCs(S) == 0) {
        /*
         * This block is being used with a fixed-step solver or it has
         * a noncontinuous sample time, so we always saturate.
         */
        for (iOutput = 0; iOutput < numOutput; iOutput++) {
            if (*uPtrs[uIdx] >= *upperLimit) {
                *y++ = *upperLimit;
            } else if (*uPtrs[uIdx] > *lowerLimit) {
                *y++ = *uPtrs[uIdx];
            } else {
                *y++ = *lowerLimit;
            }
```

```
            upperLimit += upperLimitInc;
            lowerLimit += lowerLimitInc;
            uIdx       += uInc;
        }
    } else {
        /*
         * This block is being used with a variable-step solver.
         */
        int_T *mode = ssGetModeVector(S);
        /*
         * Specify indices for each equation.
         */
        enum { UpperLimitEquation, NonLimitEquation, LowerLimitEquation };
        /*
         * Update the Mode Vector ONLY at the beginning of a MajorTimeStep
         */
        if ( ssIsMajorTimeStep(S) ) {
            /*
             * Specify the mode, ie the valid equation for each output scalar.
             */
            for ( iOutput = 0; iOutput < numOutput; iOutput++ ) {
                if ( *uPtrs[uIdx] > *upperLimit ) {
                    /*
                     * Upper limit eq is valid.
                     */
                    mode[iOutput] = UpperLimitEquation;
                } else if ( *uPtrs[uIdx] < *lowerLimit ) {
                    /*
                     * Lower limit eq is valid.
                     */
                    mode[iOutput] = LowerLimitEquation;
                } else {
                    /*
                     * Nonlimit eq is valid.
                     */
                    mode[iOutput] = NonLimitEquation;
                }
                /*
                 * Adjust indices to give scalar expansion if needed.
```

```
            */
           uIdx        += uInc;
           upperLimit += upperLimitInc;
           lowerLimit += lowerLimitInc;
       }
       /*
        * Reset index to input and limits.
        */
       uIdx        = 0;
       upperLimit = mxGetPr( P_PAR_UPPER_LIMIT );
       lowerLimit = mxGetPr( P_PAR_LOWER_LIMIT );
   } /* end IsMajorTimeStep */
   /*
    * For both MinorTimeSteps and MajorTimeSteps calculate each scalar
    * output using the equation specified by the mode vector.
    */
   for ( iOutput = 0; iOutput < numOutput; iOutput++ ) {
       if ( mode[iOutput] == UpperLimitEquation ) {
           /*
            * Upper limit eq.
            */
           *y++ = *upperLimit;
       } else if ( mode[iOutput] == LowerLimitEquation ) {
           /*
            * Lower limit eq.
            */
           *y++ = *lowerLimit;
       } else {
           /*
            * Nonlimit eq.
            */
           *y++ = *uPtrs[uIdx];
       }
       /*
        * Adjust indices to give scalar expansion if needed.
        */
       uIdx        += uInc;
       upperLimit += upperLimitInc;
       lowerLimit += lowerLimitInc;
```

```
        }
    }
} /* end mdlOutputs */
#define    MDL_ZERO_CROSSINGS
#if defined(MDL_ZERO_CROSSINGS) && (defined(MATLAB_MEX_FILE) ||
defined(NRT))
/* Function: mdlZeroCrossings ==================
 * Abstract:
 *  This will only be called if the number of nonsampled zero crossings is
 *  greater than 0 which means this block has a continuous sample time and the
 *  the model is using a variable step solver.
 *
 *  Calculate zero crossing (ZC) signals that help the solver find the
 *  exact instants at which equation switches occur:
 *
 *    if                    UpperLimit < u    then   use (1)
 *    if      LowerLimit <= u <= UpperLimit    then   use (2)
 *    if  u < LowerLimit                       then   use (3)
 *
 *  The key words are help find. There is no choice of a function that will
 *  direct the solver to the exact instant of the change. The solver will
 *  track the zero crossing signal and do a bisection style search for the
 *  exact instant of equation switch.
 *
 *  There is generally one ZC signal for each pair of signals that can
 *  switch.  The three equations above would broken into two pairs (1)&(2)
 *  and (2)&(3).  The possibility of a "long jump" from (1) to (3) does
 *  not need to be handled as a separate case.  It is implicitly handled.
 *
 *  When a ZCs are calculated, the value is normally used twice.  When it is
 *  first calculated, it is used as the end of the current time step.  Later,
 *  it will be used as the beginning of the following step.
 *
 *  The sign of the ZC signal always indicates an equation from the pair.  For
 *  S-functions, which equation is associated with a positive ZC and which is
 *  associated with a negative ZC doesn't really matter.  If the ZC is positive
 *  at the beginning and at the end of the time step, this implies that the
 *  "positive" equation was valid throughout the time step.  Likewise, if the
 *  ZC is negative at the beginning and at the end of the time step, this
```

```
 * implies that the "negative" equation was valid throughout the time step.
 * Like any other nonlinear solver, this is not fool proof, but it is an
 * excellent indicator. If the ZC has a different sign at the beginning and
 * at the end of the time step, then a equation switch definitely occured
 * during the time step.
 *
 * Ideally, the ZC signal gives an estimate of when an equation switch
 * occurred. For example, if the ZC signal is -2 at the beginning and +6 at
 * the end, then this suggests that the switch occured
 * 25% = 100% * (-2)/(-2-(+6)) of the way into the time step. It will al-
most
 * never be true that 25% is perfectly correct. There is no perfect choice
 * for a ZC signal, but there are some good rules. First, choose the ZC
 * signal to be continuous. Second, choose the ZC signal to give a monotonic
 * measure of the "distance" to a signal switch; strictly monotonic is ideal.
 */
static void mdlZeroCrossings(SimStruct *S)
{
    int_T               iOutput;
    int_T               numOutput = ssGetOutputPortWidth(S,0);
    real_T              *zcSignals = ssGetNonsampledZCs(S);
    InputRealPtrsType uPtrs       = ssGetInputPortRealSignalPtrs(S,0);
    /*
     * Set index and increment for the input signal, upper limit, and lower
     * limit parameters so that each gives scalar expansion if needed.
     */
    int_T  uIdx           = 0;
    int_T  uInc           = ( ssGetInputPortWidth(S,0) > 1 );
    real_T *upperLimit    = mxGetPr( P_PAR_UPPER_LIMIT );
    int_T  upperLimitInc = ( mxGetNumberOfElements( P_PAR_UPPER_LIM-
IT ) > 1 );
    real_T *lowerLimit    = mxGetPr( P_PAR_LOWER_LIMIT );
    int_T  lowerLimitInc = ( mxGetNumberOfElements( P_PAR_LOWER_LIM-
IT ) > 1 );
    /*
     * For each output scalar, give the solver a measure of "how close things
     * are" to an equation switch.
     */
    for ( iOutput = 0; iOutput < numOutput; iOutput++ ) {
```

```
        /*   The switch from eq (1) to eq (2)
          *
         *    if                UpperLimit < u      then    use (1)
         *    if      LowerLimit <= u <= UpperLimit      then    use (2)
          *
          *   is related to how close u is to UpperLimit.   A ZC choice
          *   that is continuous, strictly monotonic, and is
          *     u - UpperLimit
          *   or it is negative.
          */
        zcSignals[2 * iOutput] =  * uPtrs[uIdx] -  * upperLimit;
        /*   The switch from eq (2) to eq (3)
          *
          *     if        LowerLimit <= u <= UpperLimit     then    use (2)
          *     if     u < LowerLimit                        then    use (3)
          *
          *   is related to how close u is to LowerLimit.   A ZC choice
          *   that is continuous, strictly monotonic, and is
          *     u - LowerLimit.
          */
        zcSignals[2 * iOutput+1] =  * uPtrs[uIdx] -  * lowerLimit;
        /*
          * Adjust indices to give scalar expansion if needed.
          */
        uIdx        += uInc;
        upperLimit  += upperLimitInc;
        lowerLimit  += lowerLimitInc;
    }
}
#endif /* end mdlZeroCrossings */
/* Function: mdlTerminate ========================
 * Abstract:
 *     No termination needed, but we are required to have this routine.
 */
static void mdlTerminate(SimStruct * S)
{
}
#ifdef  MATLAB_MEX_FILE /* Is this file being compiled as a MEX-file? */
#include "simulink.c"        /* MEX-file interface mechanism */
```

```
#else
#include "cg_sfun.h"          /* Code generation registration function */
#endif
```

9.3.3.6　时间变化连续传递函数

S 函数例子中的 stvctf 是一个时间变化连续传递函数. 它示范了如何使用求解器,使仿真维持连贯性. 意味着尽管被积方程是变化的,模块积分保持平滑和一致的信号. 下面是其 C 代码.

```
/*
 * File : stvctf.c
 * Abstract:
 *       Time Varying Continuous Transfer Function block
 *
 *       This S-function implements a continous time transfer function
 *       whose transfer function polynomials are passed in via the input
 *       vector.  This is useful for continuous time adapative control
 *       applications.
 *
 *       This S-function is also an example of how to "use banks" to avoid
 *       problems with computing derivatives when a continuous output has
 *       discontinuities. The consistency checker can be used to verify that
 *       your S-function is correct with respect to always maintaining smooth
 *       and consistent signals for the integrators. By consistent we mean that
 *       two mdlOutput calls at major time t and minor time t are always the
 *       same. The consistency checker is enabled on the diagnostics page of the
 *       simulation parameters dialog box. The update method of this S-function
 *       modifies the coefficients of the transfer function, which cause the
 *       output to "jump." To have the simulation work properly, we need to let
 *       the solver know of these discontinuities by setting
 *       ssSetSolverNeedsReset and then we need to use multiple banks of
 *       coefficients so the coefficients used in the major time step output
 *       and the minor time step outputs are the same. In the simulation loop
 *       we have:
 *         Loop:
 *           o Output in major time step at time t
 *           o Update in major time step at time t
 *           o Integrate (minor time step):
 *               o Consistency check: recompute outputs at time t and compare
 *                 with current outputs.
```

```
 *                  o Derivatives at time t
 *                  o One or more Output,Derivative evaluations at time t+k
 *                    where k <= step_size to be taken.
 *                  o Compute state, x
 *                  o t = t + step_size
 *                End_Integrate
 *            End_Loop
 *         Another purpose of the consistency checker is used to verify that when
 *         the solver needs to try a smaller step_size that the recomputing of
 *         the output and derivatives at time t doesn't change. Step size
 *         reduction occurs when tolerances aren't met for the current step size.
 *         The ideal ordering would be to update after integrate. To achieve
 *         this we have two banks of coefficients. And the use of the new
 *         coefficients, which were computed in update, are delayed until after
 *         the integrate phase is complete.
 *
 * This block has multiple sample times and will not work correctly
 * in a multitasking environment. It is designed to be used in
 * a single tasking (or variable step) simulation environment.
 * Because this block accesses the input signal in both tasks,
 * it cannot specify the sample times of the input and output ports
 * (SS_OPTION_PORT_SAMPLE_TIMES_ASSIGNED).
 *
 * See simulink/src/sfuntmpl.doc.
 *
 * Copyright (c) 1990—1998 by The MathWorks, Inc. All Rights Reserved.
 * $Revision: 1.10 $
 */
#define S_FUNCTION_NAME  stvctf
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
/*
 * Defines for easy access to the numerator and denominator polynomials
 * parameters
 */
#define NUM(S)   ssGetSFcnParam(S, 0)
#define DEN(S)   ssGetSFcnParam(S, 1)
#define TS(S)    ssGetSFcnParam(S, 2)
#define NPARAMS 3
```

```
#define MDL_CHECK_PARAMETERS
#if defined(MDL_CHECK_PARAMETERS) && defined(MATLAB_MEX_FILE)
  /* Function: mdlCheckParameters ==================================
   * Abstract:
   *     Validate our parameters to verify:
   *      o The numerator must be of a lower order than the denominator.
   *      o The sample time must be a real positive nonzero value.
   */
  static void mdlCheckParameters(SimStruct *S)
  {
      int_T i;
      for (i = 0; i < NPARAMS; i++) {
          real_T *pr;
          int_T   el;
          int_T   nEls;
          if (mxIsEmpty(     ssGetSFcnParam(S,i)) ||
              mxIsSparse(    ssGetSFcnParam(S,i)) ||
              mxIsComplex(   ssGetSFcnParam(S,i)) ||
              ! mxIsNumeric( ssGetSFcnParam(S,i)) ) {
              ssSetErrorStatus(S,"Parameters must be real finite vectors");
              return;
          }
          pr    = mxGetPr(ssGetSFcnParam(S,i));
          nEls = mxGetNumberOfElements(ssGetSFcnParam(S,i));
          for (el = 0; el < nEls; el++) {
              if (! mxIsFinite(pr[el])) {
                  ssSetErrorStatus(S,"Parameters must be real finite vectors");
                  return;
              }
          }
      }
      if (mxGetNumberOfElements(NUM(S)) > mxGetNumberOfElements(DEN
(S)) &&
          mxGetNumberOfElements(DEN(S)) > 0  && *mxGetPr(DEN(S)) !
= 0.0) {
          ssSetErrorStatus(S,"The denominator must be of higher order than "
                             "the numerator, nonempty and with first "
                             "element nonzero");
          return;
```

```
        }
        /* xxx verify finite */
        if (mxGetNumberOfElements(TS(S)) != 1 || mxGetPr(TS(S))[0] <= 0.
0) {
            ssSetErrorStatus(S,"Invalid sample time specified");
            return;
        }
    }
  #endif /* MDL_CHECK_PARAMETERS */
  /* Function: mdlInitializeSizes ===================
  * Abstract:
  *     The sizes information is used by Simulink to determine the S-function
  *     block's characteristics (number of inputs, outputs, states, etc.).
  */
  static void mdlInitializeSizes(SimStruct *S)
  {
      int_T nContStates;
      int_T nCoeffs;
      /* See sfuntmpl.doc for more details on the macros below. */

      ssSetNumSFcnParams(S, NPARAMS);  /* Number of expected parameters.
*/
  #if defined(MATLAB_MEX_FILE)
      if (ssGetNumSFcnParams(S) == ssGetSFcnParamsCount(S)) {
          mdlCheckParameters(S);
          if (ssGetErrorStatus(S) != NULL) {
              return;
          }
      } else {
          return; /* Parameter mismatch will be reported by Simulink. */
      }
  #endif
      /*
       * Define the characteristics of the block:
       *
       *    Number of continuous states:    length of denominator - 1
       *    Inputs port width                 2 * (NumContStates+1) + 1
       *    Output port width                 1
       *    DirectFeedThrough:     0 (Although this should be computed.
```

```
 *                               We'll assume coefficients entered
 *                                            are strictly proper).
 *   Number of sample times:              2 (continuous and discrete)
 *   Number of Real work elements:       4 * NumCoeffs
 *                                 (Two banks for num and den coeff's:
 *                                  NumBank0Coeffs
 *                                  DenBank0Coeffs
 *                                  NumBank1Coeffs
 *                                  DenBank1Coeffs)
 *   Number of Integer work elements: 2 (indicator of active bank 0 or 1
 *                                    and flag to indicate when banks
 *                                    have been updated).
 *
 * The number of inputs arises from the following:
 *   o 1 input (u)
 *   o the numerator and denominator polynomials each have NumContStates+1
 *     coefficients
 */
nCoeffs      = mxGetNumberOfElements(DEN(S));
nContStates = nCoeffs - 1;
ssSetNumContStates(S, nContStates);
ssSetNumDiscStates(S, 0);
if (! ssSetNumInputPorts(S, 1)) return;
ssSetInputPortWidth(S, 0, 1 + (2 * nCoeffs));
ssSetInputPortDirectFeedThrough(S, 0, 0);
ssSetInputPortSampleTime(S, 0, mxGetPr(TS(S))[0]);
ssSetInputPortOffsetTime(S, 0, 0);
if (! ssSetNumOutputPorts(S,1)) return;
ssSetOutputPortWidth(S, 0, 1);
ssSetOutputPortSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
ssSetOutputPortOffsetTime(S, 0, 0);
ssSetNumSampleTimes(S, 2);
ssSetNumRWork(S, 4 * nCoeffs);
ssSetNumIWork(S, 2);
ssSetNumPWork(S, 0);
ssSetNumModes(S, 0);
ssSetNumNonsampledZCs(S, 0);
/* Take care when specifying exception free code - see sfuntmpl.doc */
ssSetOptions(S, (SS_OPTION_EXCEPTION_FREE_CODE));
```

```
} /* end mdlInitializeSizes */
/* Function: mdlInitializeSampleTimes ================
 * Abstract:
 *       This function is used to specify the sample time(s) for the
 *       S-function.  This S-function has two sample times.  The
 *       first, a continous sample time, is used for the input to the
 *       transfer function, u.  The second, a discrete sample time
 *       provided by the user, defines the rate at which the transfer
 *       function coefficients are updated.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    /*
     * the first sample time, continuous
     */
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
    /*
     * the second, discrete sample time, is user provided
     */
    ssSetSampleTime(S, 1, mxGetPr(TS(S))[0]);
    ssSetOffsetTime(S, 1, 0.0);
} /* end mdlInitializeSampleTimes */
#define MDL_INITIALIZE_CONDITIONS
/* Function: mdlInitializeConditions ====================
 * Abstract:
 *       Initalize the states, numerator and denominator coefficients.
 */
static void mdlInitializeConditions(SimStruct *S)
{
    int_T   i;
    int_T   nContStates = ssGetNumContStates(S);
    real_T  *x0         = ssGetContStates(S);
    int_T   nCoeffs     = nContStates + 1;
    real_T  *numBank0   = ssGetRWork(S);
    real_T  *denBank0   = numBank0 + nCoeffs;
    int_T  *activeBank  = ssGetIWork(S);
    /*
     * The continuous states are all initialized to zero.
```

```
     */
    for (i = 0; i < nContStates; i++) {
        x0[i]         = 0.0;
        numBank0[i] = 0.0;
        denBank0[i] = 0.0;
    }
    numBank0[nContStates] = 0.0;
    denBank0[nContStates] = 0.0;
    /*
     * Set up the initial numerator and denominator.
     */
    {
        const real_T *numParam    = mxGetPr(NUM(S));
        int           numParamLen = mxGetNumberOfElements(NUM(S));
        const real_T *denParam    = mxGetPr(DEN(S));
        int           denParamLen = mxGetNumberOfElements(DEN(S));
        real_T        den0        = denParam[0];
        for (i = 0; i < denParamLen; i++) {
            denBank0[i] = denParam[i] / den0;
        }
        for (i = 0; i < numParamLen; i++) {
            numBank0[i] = numParam[i] / den0;
        }
    }
    /*
     * Normalize if this transfer function has direct feedthrough.
     */
    for (i = 1; i < nCoeffs; i++) {
        numBank0[i] -= denBank0[i] * numBank0[0];
    }
    /*
     * Indicate bank0 is active (i.e. bank1 is oldest).
     */
    *activeBank = 0;
} /* end mdlInitializeConditions */
/* Function: mdlOutputs ========================================
 * Abstract:
 *     The outputs for this block are computed by using a controllable state-
 *     space representation of the transfer function.
```

```
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
    if (ssIsContinuousTask(S,tid)) {
        int                 i;
        real_T               *num;
        int                 nContStates = ssGetNumContStates(S);
        real_T               *x          = ssGetContStates(S);
        int_T               nCoeffs      = nContStates + 1;
        InputRealPtrsType uPtrs         = ssGetInputPortRealSignalPtrs(S,0);
        real_T               *y          = ssGetOutputPortRealSignal(S,0);
        int_T                *activeBank = ssGetIWork(S);
        /*
         * Switch banks we've updated them in mdlUpdate and we're no longer
         * in a minor time step.
         */
        if (ssIsMajorTimeStep(S)) {
            int_T *banksUpdated = ssGetIWork(S) + 1;
            if (*banksUpdated) {
                *activeBank = !(*activeBank);
                *banksUpdated = 0;
                /*
                 * Need to tell the solvers that the derivatives are no
                 * longer valid.
                 */
                ssSetSolverNeedsReset(S);
            }
        }
        num = ssGetRWork(S) + (*activeBank) * (2*nCoeffs);
        /*
         * The continuous system is evaluated using a controllable state space
         * representation of the transfer function.  This implies that the
         * output of the system is equal to:
         *
         *      y(t) = Cx(t) + Du(t)
         *           = [ b1 b2 ... bn]x(t) + b0u(t)
         *
         * where b0, b1, b2, ... are the coefficients of the numerator
         * polynomial:
```

```
          *
          *     B(s) = b0 s^ n + b1 s^ n-1 + b2 s^ n-2 + ... + bn-1 s +
bn
          */
         *y = *num++ * (*uPtrs[0]);
         for (i = 0; i < nContStates; i++) {
             *y += *num++ * *x++;
         }
     }
} /* end mdlOutputs */
#define MDL_UPDATE
/* Function: mdlUpdate ==============
 * Abstract:
 *      Every time through the simulation loop, update the
 *      transfer function coefficients. Here we update the oldest bank.
 */
static void mdlUpdate(SimStruct *S, int_T tid)
{
    if (ssIsSampleHit(S, 1, tid)) {
        int_T              i;
        InputRealPtrsType uPtrs        = ssGetInputPortRealSignalPtrs(S,0);
        int_T              uIdx        = 1;/* 1st coeff is after signal input */
        int_T              nContStates = ssGetNumContStates(S);
        int_T              nCoeffs     = nContStates + 1;
        int_T              bankToUpdate = ! ssGetIWork(S)[0];
        real_T     *num    = ssGetRWork(S)+bankToUpdate * 2 * nCoeffs;
        real_T             *den        = num + nCoeffs;
        real_T             den0;
        int_T              allZero;
        /*
         * Get the first denominator coefficient.  It will be used
         * for normalizing the numerator and denominator coefficients.
         *
         * If all inputs are zero, we probably could have unconnected
         * inputs, so use the parameter as the first denominator coefficient.
         */
        den0 = *uPtrs[uIdx+nCoeffs];
        if (den0 == 0.0) {
            den0 = mxGetPr(DEN(S))[0];
```

```
}
/*
 * Grab the numerator.
 */
allZero = 1;
for (i = 0; (i < nCoeffs) && allZero; i++) {
    allZero &= *uPtrs[uIdx+i] == 0.0;
}
if (allZero) { /* if numerator is all zero */
    const real_T *numParam    = mxGetPr(NUM(S));
    int_T      numParamLen = mxGetNumberOfElements(NUM(S));
    /*
     * Move the input to the denominator input and
     * get the denominator from the input parameter.
     */
    uIdx += nCoeffs;
    num += nCoeffs - numParamLen;
    for (i = 0; i < numParamLen; i++) {
        *num++ = *numParam++ / den0;
    }
} else {
    for (i = 0; i < nCoeffs; i++) {
        *num++ = *uPtrs[uIdx++] / den0;
    }
}
/*
 * Grab the denominator.
 */
allZero = 1;
for (i = 0; (i < nCoeffs) && allZero; i++) {
    allZero &= *uPtrs[uIdx+i] == 0.0;
}
if (allZero) {  /* If denominator is all zero. */
    const real_T *denParam    = mxGetPr(DEN(S));
    int_T          denParamLen = mxGetNumberOfElements(DEN(S));
    den0 = denParam[0];
    for (i = 0; i < denParamLen; i++) {
        *den++ = *denParam++ / den0;
    }
```

```
        } else {
            for (i = 0; i < nCoeffs; i++) {
                *den++ = *uPtrs[uIdx++] / den0;
            }
        }
        /*
         * Normalize if this transfer function has direct feedthrough.
         */
        num = ssGetRWork(S) + bankToUpdate * 2 * nCoeffs;
        den = num + nCoeffs;
        for (i = 1; i < nCoeffs; i++) {
            num[i] -= den[i] * num[0];
        }
        /*
         * Indicate oldest bank has been updated.
         */
        ssGetIWork(S)[1] = 1;
    }
} /* end mdlUpdate */
#define MDL_DERIVATIVES
/* Function: mdlDerivatives =====================
 * Abstract:
 *      The drivatives for this block are computed by using a controllable
 *      state-space representation of the transfer function.
 */
static void mdlDerivatives(SimStruct *S)
{
    int_T           i;
    int_T           nContStates = ssGetNumContStates(S);
    real_T          *x          = ssGetContStates(S);
    real_T          *dx         = ssGetdX(S);
    int_T           nCoeffs     = nContStates + 1;
    int_T           activeBank  = ssGetIWork(S)[0];
    const real_T    *num        = ssGetRWork(S) + activeBank * (2 *
nCoeffs);
    const real_T    *den        = num + nCoeffs;
    InputRealPtrsType uPtrs     = ssGetInputPortRealSignalPtrs(S,0);
    /*
     * The continuous system is evaluated using a controllable state-space
```

```
    * representation of the transfer function.  This implies that the
    * next continuous states are computed using:
    *
    *     dx = Ax(t) + Bu(t)
    *        = [-a1 -a2 ... -an] [x1(t)] + [u(t)]
    *          [ 1   0  ...   0] [x2(t)] + [0]
    *          [ 0   1  ...   0] [x3(t)] + [0]
    *          [ .   .  ...   .]   .     +  .
    *          [ .   .  ...   .]   .     +  .
    *          [ .   .  ...   .]   .     +  .
    *          [ 0   0  ... 1 0] [xn(t)] + [0]
    *
    * where a1, a2, ... are the coefficients of the numerator polynomial:
    *
    *     A(s) = s^n + a1 s^n-1 + a2 s^n-2 + ... + an-1 s + an
    */
   dx[0] = -den[1] * x[0] + *uPtrs[0];
   for (i = 1; i < nContStates; i++) {
       dx[i] = x[i-1];
       dx[0] -= den[i+1] * x[i];
   }
} /* end mdlDerivatives */
/* Function: mdlTerminate =====================
 * Abstract:
 *     Called when the simulation is terminated.
 *     For this block, there are no end of simulation tasks.
 */
static void mdlTerminate(SimStruct *S)
{
} /* end mdlTerminate */

#ifdef  MATLAB_MEX_FILE     /* Is this file being compiled as a MEX-file?
*/
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

9.3.4 使用 Function-Call 子系统

可以创建一个可触发子系统，它的执行取决于一个 S 函数的内部逻辑而不是信号的

值.一个如此设定的子系统被称为 function-call 子系统.要实现一个 function-call 子系统，按如下的步骤进行：

1）在 Trigger 模块中，选择 function-call 作为 Trigger type 参数.

2）在 S 函数中，使用 ssCallSystemWithTid 宏去调用可触发子系统.

3）在模型中，将 S 函数模块的输出直接连到触发端口.

Function-call 子系统并不是直接被 Simulink 执行.S 函数确定何时执行该子系统.当子系统执行完毕，控制返回给 S 函数.

Function-call 子系统只能够连接到已经正确地配置了可以接受它们的 S 函数.要设定一个 S 函数可以调用一个 function-call 子系统，按如下步骤进行：

1）在 mdlInitializeSampleTimes 中指定哪些元素将用来执行 function-call 系统.如：

```
ssSetCallSystemOutput(S,0); /* call on 1st element */
ssSetCallSystemOutput(S,2); /* call on 3rd element */
```

2）在适当的 mdlOutputs 或 mdlUpdates 子程序中执行该子系统.例如：

```
static void mdlOutputs(...)
{
  if (((int) * uPtrs[0]) % 2 == 1) {
    if (! ssCallSystemWithTid(S,0,tid)) {
      /* Error occurred, which will be reported by Simulink */
      return;
    }
  } else {
    if (! ssCallSystemWithTid(S,2,tid)) {
      /* Error occurred, which will be reported by Simulink */
      return;
      }
    }
    ...
}
```

在 sfun_fcncall.c 中演示了一个 S 函数被设定成可执行 function-call 子系统.Function-call 子系统通常被用于 Stateflow 模块.

9.3.5 S 函数类型

（1）非嵌入 S 函数

非嵌入 S 函数是一个 C MEX S 函数，在 Simulink 和 RTW 中同样对待.一般情况下，以前是根据 S 函数 API 来实现算法的.而 Simulink 和 RTW 调用 S 函数子程序是在模型执行的合适的点上.每个非嵌入 S 函数模块实例都需要重要的内存和计算资源.这种综合 Simulink 和 RTW 的算法子程序，一般在原型设计阶段使用，此时效率并不重要.其优点是带来了快速改变模型参数和结构的能力.

写非嵌入式 S 函数不包含 TLC 代码.非嵌入式 S 函数是 RTW 的缺省情况，但在

模型中创建了 S 函数，在模型的仿真参数对话框的 RTW 页面中按 Build 之前，无需其它准备工作.

对于非嵌入式 S 函数的执行，需要 C MEX S 函数的源代码(sfunction. c).

(2) 包装 S 函数

包装 S 函数对于连接手写代码或包装在一些程序中的大算法是非常理想的. 这种情况下，通常这些程序驻留在与 C MEX S 函数分离的模块中. S 函数模块通常包含对这些程序的一些调用. 由于 S 函数不包含这些算法的任何部分，仅调用其代码，因此，叫做包装 S 函数. 除了包装 C MEX S 函数包装外，还需要创建 TLC 包装来实现 S 函数. TLC 包装与 S 函数包装一样，其中包含有算法程序.

对于嵌入调用算法(C 语言函数)的包装 S 函数的执行，需要一个 sfunction. tlc 文件.

(3) 完全嵌入 S 函数

完全嵌入 S 函数，将以一种与内置模块没有区别的方式，在 Simulink 和 RTW 中创建算法(模块). 一般地，完全嵌入 S 函数需要执行算法两次：一是为 Simulink(C MEX S 函数)，二是为 RTW(TLC 文件). TLC 文件的复杂性取决于算法的复杂性，以及产生的代码要达到的效率水平. TLC 文件结构有简单有复杂.

对于完全嵌入式 S 函数的执行，需要一个 sfunction. tlc 文件.

对于使用 RTW 的 S 函数子程序的完全嵌入式 S 函数，需要放置 mdlRTW 子程序于 S 函数 MEX 文件 sfunction. c 中. mdlRTW 子程序使得用户可以放置信息于 model. rtw 中，这是一个在产生代码和导入非可调参数时，在执行 sfunction. tlc 之前，用目标语言编译器处理的文件.

[G e n e r a l I n f o r m a t i o n]
书名＝应用ＭＡＴＬＡＢ建模与仿真
作者＝
页数＝３９８
ＳＳ号＝０
出版日期＝